Artificial Intelligence and Law
https://doi.org/10.1007/s10506-025-09456-8

ORIGINAL RESEARCH

Automated consistency analysis for legal contracts

Alan Khoja[2] · **Martin Kölbl**[1] · **Stefan Leue**[1] · **Rüdiger Wilhelmi**[2]

Accepted: 3 April 2025

Abstract

Business contracts, particularly sale and purchase agreements, often contain a large number of clauses and are correspondingly long and complex. In practice, it is therefore a great challenge to keep track of their legal context and to identify and avoid inconsistencies in such contracts. Against this background, we describe a method and tool called *ContractCheck* which allows for the consistency analysis of legal contracts, in particular share purchase agreements (SPAs). In order to identify the concepts that are relevant for an analysis we define an ontology for SPAs. The analysis is, then, based on an encoding of the preconditions for the execution of the clauses of an SPA, as well as on a set of proposed consistency constraints formalized using decidable fragments of first-order logic (FOL). Based on the ontology for SPAs, textual SPAs are first encoded in a structured natural language format that we refer to as "blocks". *ContractCheck* interprets these blocks and constraints and translates them into assertions formulated in FOL. It then invokes a Satisfiability Modulo Theory (SMT) solver in order to check the executability of a considered contract, either by providing a satisfying model, or by proving the existence of conflicting clauses that prevent the contract from being executed. We illustrate the application of *ContractCheck* to concrete SPAs, including one example of an SPA of realistic size and complexity, and conclude by suggesting directions for future research.

Keywords Logic and law · Contracts · Consistency · SMT solving · Mergers and acquisitions · Share purchase agreement

✉ Stefan Leue
Stefan.Leue@uni-konstanz.de

Alan Khoja
Alan.Khoja@uni-konstanz.de

Martin Kölbl
Martin.Koelbl@uni-konstanz.de

Rüdiger Wilhelmi
Ruediger.Wilhelmi@uni-konstanz.de

[1] Department of Computer Science, University of Konstanz, Konstanz, Germany

[2] Department of Law, University of Konstanz, Konstanz, Germany

Published online: 13 May 2025

1 Introduction

1.1 Motivation and overview

Contracts are essential in business. They allow the contracting parties to arrange their legal relationships by giving legal effect to their common will and establishing mutual claims. A prominent example is the purchase of a company in a share purchase agreement (SPA). Like any contract of sale, an SPA must contain at least the indispensable *essentialia negotii*: the purchaser and the seller, as well as the subject matter of the purchase and the purchase price to be paid. In practice, SPAs regulate all relevant legal issues in the contract and exclude references to statutory law as far as possible. As a result, SPAs have a very local semantics, based almost entirely on the contractual obligations agreed in the SPA (Oetker and Maultzsch 2018).

Contracts, and especially SPAs, are often very long and complex. This is due to the large number and complexity of the issues to be addressed. In addition, a large number of people are often involved in drafting contract texts. Furthermore, the negotiation and the drafting process may extend over a long period of time and involve a large number of changes to the draft. Length, complexity and frequent changes make a contract prone to errors and inconsistencies, such as incorrect references, missing essential elements and unfulfillable or unenforceable claims. Inconsistencies in the form of missing essentials can be found by simple syntactic analysis. It is much more complex to find inconsistencies in the dynamics of multiple claims. Claims should not contradict each other and should be fulfillable and enforceable in the context of the legal facts described in the contract. Also, the combination of several due dates may be unexpectedly restricted by a statute of limitations stated in the contract.

For instance, assume that an SPA contains a warranty claim that must be asserted within 14 days after the closing on day 28, then subsequent performance must be made within 28 days, and otherwise damages must be paid within another 14 days. Assume further that the contract contains a provision that warranty claims are limited to 42 days after closing. In this example, the warranty clause provides that performance may continue until day 84, while the limitation period ends on day 70. This means that there are inconsistencies in the timing of the SPA.

We pursue a research agenda that aims at developing analysis methods for the identification of logical inconsistencies, such as the ones described above, in contract texts, with a specific focus on SPAs. It is our objective to develop automated methods and tools that do not require input from the user, for instance guidance of mathematical proving or logical reasoning. We combine this objective with the goal to permit reasoning about a fairly wide range of data domains, for instance natural or rational numbers, arithmetic expressions and uninterpreted functions representing uninterpreted logical predicates. This allows us to reason about, e.g., prices, dates, timing, interest rates, compensation amounts and ownership properties. The identification of inconsistencies in contract texts is an objective per se. However, it also enables determining the executability of a contract in terms of sequences of claim performances, or as a whole. As a use case for the developed method and tool we

envision support for legal actors during contract drafting and negotiation, among others.

In this paper we propose a method and tool called *ContractCheck*, designed to formally model and automatically analyze a given SPA. From a technical perspective, the *ContractCheck* approach is rooted in concepts from symbolic artificial intelligence, such as symbolic logic reasoning and automated theorem proving, and inspired by experiences with the use of formal methods in the analysis of software and system designs. We maintain that automated formal analysis enables the localization of erroneous or inconsistent contractual text by automated logical reasoning, thereby significantly improving the quality of the contract document. Experience shows that the adoption of formal analyses in practice is greatly enhanced by a) hiding the formality in the analysis method, i.e., not confronting the end user with the need to express desired properties or the system model using a formal notation, and b) the use of fully automated analysis methods that do not impose the burden of driving the analysis process by providing manual guidance on the user as it would, for instance, be required when using an interactive theorem prover. These insights have guided our development of the technical aspects of *ContractCheck*, in particular the automated form of logical reasoning that *ContractCheck* relies on.

Against this backdrop, we propose to base the analysis of consistency properties in SPAs on a formalization of the claims that they encompass using decidable fragments of First-Order Logic (FOL), and to perform the analysis using Satisfiability Modulo Theory (SMT) solving techniques (Kroening and Strichman 2016; Clarke et al. 2018; Bjørner 2009). More precisely, we consider an SPA to be a collection of claims. We define a weakest precondition style semantics for the satisfaction of claims that is based on enabling conditions for their executability, such as the timing constraints used in the above example. Based on this formalization, we define two types of logical consistency analysis questions, namely *Analyis I*, is every claim in the contract executable, and *Analysis II*, is there a feasible execution of the SPA. Answers to these questions rely on determining the satisfiability of the conjunction of the set of first-order formulae that formalize the claims and the contract execution. The decision procedures required to analyze satisfiability of these formulae are efficiently implemented in various SMT solving tools, such as the Z3 solver (de Moura and Bjørner 2008; Bjørner 2009) developed at Microsoft Research, which we use in our analysis.

1.2 Structure of the paper

After discussing some preliminaries in Sect. 2, we first develop an ontological metamodel for the relevant entities of an SPA and present this as a class diagram from the Unified Modeling Language (UML) (Object Management Group 2017) in Sect. 3. Next, we define a semantics for the executability of clauses in an SPA using decidable fragments of quantifier-free FOL in Sect. 4. The formalization of the various automated consistency analyses performed by *ContractCheck* is presented in Sect. 5. Using the running example of the sale of a pretzel bakery, we then illustrate the automatic consistency analysis of a given SPA using the SMT technology invoked

by *ContractCheck* in Sect. 6. We demonstrate the ability of *ContractCheck* to detect and explain inconsistencies in a more complex, realistic SPA drawn from the literature in Sect. 7. We discuss threats to the validity of our results in Sect. 8. Finally, we conclude and outline future research in Sect. 9.

1.3 Scope and methodological considerations

In this paper, we develop the conceptual basis for *ContractCheck* using SPAs under German law, but this does not preclude adaptation to contracts with a different subject matter or under other jurisdictions, as they may be more complex but basically have a comparable structure with similar elements.

Looking at examples of SPAs, it is noticeable that they often consist of very similar blocks of text, differing only in certain parameters, such as the agreed price. In the case of international company acquisitions, it is estimated that about half of the text of the provisions is changed little or not at all (Coates 2016). In order to achieve a formal logical representation of the SPA, as part of our approach and tool we provide a library of parameterized Structured English text blocks that can be freely combined and used to compose the text of the contract. These blocks allow greater flexibility than conventional approaches, where contract creation is dialogue-driven and based on decision trees. Each of these blocks has a formal semantics expressed by formulae from a decidable fragment of FOL. The conjunction of these conditions then constitutes the logical representation of the contract. The analysis method that we describe in this paper generalizes to the class of all concrete contracts that can be formulated by composing and parameterizing the provided text blocks.

1.4 Approach

Figure 1 sketches our proposed approach towards the analysis of legal contracts. First, we present a `UML Ontology` which provides a vocabulary of entities from the domain of SPAs as well as a representation of their relationships. Based on this ontology, the *ContractCheck* tool provides a number of different parameterized Structured English text skeletons, referred to as `blocks`, that can be used to formally capture properties of the different clauses in an SPA. From the perspective

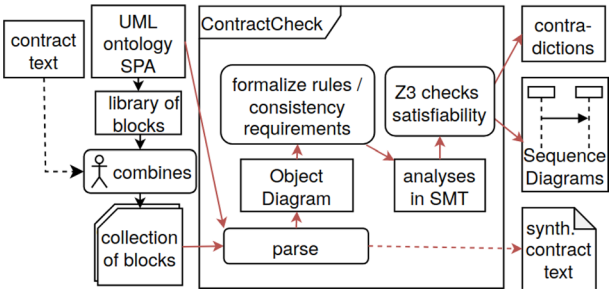


Fig. 1 The *ContractCheck* approach towards automated contract analysis

of contract generation, the idea underlying this approach is that users are provided a collection of `blocks`, each representing a different type of contract clause, that they can then compile to form a textual contract document. The resulting collection of `blocks` can then be automatically be automatically analyzed by the *ContractCheck* tool. Notice that in our experiments we do not (yet) take advantage of this contract generation feature. Instead, we are taking existing contract texts and translate them manually into collections of blocks for later analysis.

ContractCheck implements an automated analysis workflow as indicated by the red arrows in Fig. 1. For analysis purposes, *ContractCheck* parses the blocks, determines the entities and relationships using the UML ontology and generates an internal representation of the considered SPA in the form of an UML `Object Diagram`. The syntactic correctness of the considered SPA is determined by checking the consistency of the object diagram with the UML-defined ontology. For dynamic consistency checks, several consistency `analyses` are generated. In order to prepare the formal analysis, the object diagram is translated into *SMT-LIB* (Barrett et al. 2021) format, which the `Z3` SMT solver then checks for satisfiability, thereby determining logical consistency of the clauses contained in the considered SPA. The results obtained during the SMT solving are presented either by flagging the blocks that are determined to contradict each other (Analysis I), or by sequence diagrams representing possible SPA executions in case a contradiction-free execution of the SPA is possible (Analysis II).

1.5 Related work

We review publications that address the logical modeling and analysis of legal artifacts. We consider the verification of smart contracts, c.f. Braegelmann and Kaulartz (2019), Paal and Fries (2019), which are effectively described by executable program code, to be outside the scope of this paper, since their analysis is more akin to program analysis.

Legal modeling and analysis

Several meta-models of different legal domains aiming to describe legal entities and their relationships have been proposed. A first approach towards representing contracts with UML has been proposed in Engers et al. (2001). A multi-level hierarchical ontology relying on UML class diagrams in order to support the modeling of contracts is presented in Kabilan and Johannesson (2003). A modeling of contracts as business processes has been proposed in Kabilan (2005), Wan and Singh (2005). In Desai et al. (2008), a business process is translated into state machines to check whether it is always beneficial for the contracting parties to fulfill their claims during the execution of the contract. Dynamic Condition Response Graphs (Hildebrandt et al. 2012) (DCR graphs) provides a graph-based language to model the dynamic dependencies and interactions between different legal conditions and responses along legal workflows. Temporal and logical properties of the event relations can be checked using formal analyses, including explicit-state model checking (Clarke et al. 2018).

LegalRuleML (Palmirani et al. 2011; OASIS Standard 2021; Grupp 2018) is a specialization of the general language *RuleML* (Boley et al. 2010) for expressing relationships between legal entities by means of rules. We are not aware of any extension of LegalRuleML to analyze contract executions for inconsistencies. The Contract Specification Language (CSL) is a modeling language that represents claims as actions in a contract. In Hvitved et al. (2012), the execution of a given CSL expression is computed as a sequence of actions, which are then analyzed to determine whether any specified obligations are met. In Henglein et al. (2020), a contract is interpreted and analyzed as a composition of commitments. In Madaan et al. (2014), the natural language sentences of an e-contract are translated into a dependency graph in order to check whether individual provisions contradict each other.

Deontic logic (Von Wright 1951; McNamara and Van De Putte 2022) is a family of modal logics designed to express the semantics of claims with operators for the modalities of obligation, permission, and prohibition. A Deontic Propositional Logic and its axiomatization are defined in Castro and Maibaum (2007). A tableau calculus for this logic is defined in Castro and Maibaum (2008). Tableau-based decision procedures for a class of logics encompassing temporal and deontic modalities are proposed in Balbiani et al. (2009). These analyses find inconsistencies with respect to deontic or temporal modalities within a contract, but do not calculate contract executions witnessing these inconsistencies.

The Contract Language *CL* (Prisacariu and Schneider 2007; Prisacariu and Schneider 2012) encompasses deontic as well as dynamic propositional logic. In Pace et al. (2007), a contract given in *CL* is translated into a transition system, which is then analyzed by the model checker NuSMV for inconsistencies. In Camilleri and Schneider (2017), a *C-O diagram* (Martínez et al. 2010), which is a graphical extension of *CL*, is expressed by state machines and analyzed using the real-time model checker UPPAAL. The analyses find syntactic inconsistencies and prove a dynamic inconsistency by a single contract execution. The approach requires an expert to encode a contract as a *C-O diagram*. An extension of *CL* to *RCL* in Mura and Bonifácio (2015) and Bonifácio and Mura (2021) proposes reasoning about the persons between whom claims exist using the *RECALL* tool.

Another system supporting a deontic logic-based approach is the Linear Time Temporal Logic (LTL)-based language *FL* (Gorín et al. 2010; Gorín 2011) together with the model checker *FormaLex* (Gorín 2011; Faciano et al. 2017).

Schumann and Gómex (2024) review research on consistency analysis in regulatory documents. They observe that the most frequently used class of techniques for detecting inconsistencies in the studies that they analyze are formal verification methods. In this context, Mitra et al. (2018) describe consistency analysis of business workflow rules based on an encoding of these rules in the SMT-LIB formalism and the performance of SMT solving on this encoding. Unlike our work, this work does not analyze the consistency of legal contracts.

Programming and domain specific languages Catala (Merigoux et al. 2021) is a functional programming language developed by Microsoft. It is specifically designed for use in a legal context and is intended to be used to automatically interpret and apply laws and regulations. *Flint/eFlint* (Van Doesburg et al. 2016; van

Binsbergen et al. 2020) is a domain-specific language aiming at the formalization of normative legal texts. It is based on the framework of legal fundamentals by Kennedy and Fisher (2007). The Flint/eFlint models can be executed and, to a limited extent, analyzed. *L4* (Governatori and Wong 2023), also known as Governatori's domain-specific language for the legal domain, is a language that allows the expression of legal rules in a simple and structured way, using the syntax "if precondition then conclusion". It is intended to make it easier for legal experts to express legal rules in a way that can be understood by computers. *Stipula* (Crafa et al. 2023) is a domain-specific language that allows for the creation of software legal contracts and smart legal contracts, focusing on deontic concepts such as permissions, prohibitions, obligations, assets exchanges, escrows and securities.

1.6 Comparison to related work

The logical analysis approach that we propose in this paper focuses on the use of decidable fragments of FOL in the formalization and the use of decision procedures for these fragments, implemented in SMT technology, to detect inconsistencies in SPAs. While many of the approaches mentioned above do have the ability to detect inconsistencies in the execution of actions, they do not have the ability to also produce models that reveal reasons for inconsistencies that lie in the data, which is an aspect that we focus on. For instance, in comparison to the cited approaches that use the UPPAAL real-time model checking tool, we are able to reason at the same time about inconsistencies in the timing of the execution of claims and in the constraints on numerical values, such as purchase prices and warranty claims.

Only finite executions are of interest in an SPA, which means that LTL-style model checking capabilities of infinite behaviors are not required for the purpose of our consistency analysis.[1] The types of analysis performed by *FormaLex* are based on standard LTL model checking and encompass a temporal interpretation of deontic modalities, such as obligation. The objectives of this work differ from ours in that the analysis of domain data is not addressed in *FormaLex*, while we are not focusing on reasoning about deontic modalities.

A further difference compared to the cited works lies in the ability of *ContractCheck* to prioritize primary over secondary claims, an important legal concept in SPAs, during Analysis II, which allows for a very meaningful type of executability analysis. The above cited approaches are only able to logically link different claims without this type of prioritization.

Comparison to legal reasoning using deontic logics Reasoning based on deontic logics focuses on the modalities of obligation, permission, and prohibition, which are ideally suited to reason about properties of claims in legal contracts that need to be cast in terms of these modalities. Reasoning in these logics means to derive inferences from the modalities obligatory, impermissible, optional, non-optional,

[1] Notice that we are not interested in, for instance, proving that an SPA does not include non-progress cycles. Consequently, we do not need to reason about liveness properties and can limit our analyses to safety properties, c.f. Alpern and Schneider (1987).

permissible and obligatory in the "deontic hexagon" (McNamara and Van De Putte 2022). Depending on the choice of a syntactic fragment and the assumed semantics, deontic logics may be prone to paradoxes, which limits the possibility of consistency checking, although numerous fragments with decidable, consistent semantics have been defined, see for instance Prisacariu and Schneider (2012).

While the logic of claims and the possible sequences of their performance, which we refer to as contract executions, are a central concept in our paper, we are not interested in reasoning in terms of the above mentioned deontic modalities. Our approach bears similarities with semantic reasoning about programs, where contract claims play the role of program statements. We reason whether starting from an initial configuration of the contractual situation, due to the satisfaction of some logical condition (precondition), the performance of a claim is possible, and which logical result conditions (postcondition) its execution entails. The predicates that we use as pre- and postconditions are formulated as FOL formulae over various decidable theories, such as integer and real arithmetic. This allows us to specify conditions on domain-specific values, such as on due dates for claim performance, purchase prices or warranty compensations. We only choose fragments of FOL that are decidable and for which efficient decision procedures, implemented in SMT solvers such as Z3, are available. These fully automated solvers are capable of returning models that provide evidence for why a contractual situation is inconsistent.

In conclusion, our approach differs from other analysis approaches relying on deontic logics in that it combines reasoning on possible contract executions, checking the consistency of domain-specific predicates, returns explanations for inconsistencies using domain-specific values, and is availing itself to efficient automated analysis through practical SMT-solving.

1.7 Contributions

This paper builds on results that appeared in Khoja et al. (2022a, 2022b). It presents an enhanced description of the method, a more comprehensive formalization and a more realistic case study. In particular, the main contributions of the work documented in this paper are as follows:

1. We propose an ontology for SPAs that we document using the UML.
2. We provide a logical formalization of the semantics of SPAs using decidable fragments of FOL. In doing so, we contribute to the extensive research area dealing with the use of formal logics in the representation of legal artefacts.
3. We define a set of consistency analyses that are applicable to SPAs.
4. We describe a tool that uses a collection of parameterized natural language building blocks from which textual SPAs can be composed. The possible synthesis of contract text is not considered in this paper.

5. We present the prototype tool called *ContractCheck*, that uses SMT and satisfiability core technology to perform the consistency analyses and to produce diagnostic information explaining identified inconsistencies.
6. We illustrate the application of this approach to an SPA of a complex example from legal practice to show that the method is effective and efficient.

2 Preliminaries

We introduce key legal terms, the modeling language used to describe the company sale contract, and the logic we use to formalize a contract.

2.1 Basic legal terms

Even if statutory provisions are largely excluded in an SPA, they are still relevant for the drafting and interpretation of the agreement (Wilhelmi 2024a). This is because the contractual provisions are based on the underlying legal system and its systematics. In particular, the SPA that we use to illustrate our approach contains terms that can only be understood by reference to the legal provisions and concepts of the German Civil Code (BGB).

Against this backdrop, we explain some basic legal concepts of private law generally used in contracts, and in SPAs in particular. These concepts regard, first of all, the distinction between legal subjects and legal objects, the relationships between legal subjects and legal objects and the possibility to change legal positions.

Legal subjects are, on the one hand, *natural persons*, i.e. human beings, and, on the other hand, *legal persons*, in private law in particular associations and other corporate bodies such as the public or private limited company. Only they can be the bearers of rights and obligations and, in particular, be the parties to a contract (of sale).

Legal objects may be the subject of rights of control and use as well as rights of disposal of legal subjects. In particular, they include *things*, i.e., physical objects, but also *rights*, such as *claims* of one legal subject against another legal subject or company shares, and other assets, such as intangible assets.

Relations between legal subjects are, in particular, *claims*, i.e., the *right* of one *legal subject* to demand an act or omission from another *legal subject*, corresponding with the *obligation* of the other *legal subject* to this act or omission. In contracts, a distinction can be made between primary and secondary claims. *Primary claims* are the original claims that exist even if the contract is performed without disturbance, while *secondary claims* only arise if the performance of the contract is disturbed and primary claims or other promises are breached. Such promises can be *warranties* regarding the existence or non-existence of circumstances. In case of a breach, they can provide a claim for subsequent *performance*, *restitution* or *compensation*.

Relations between legal subjects and legal objects are also possible. A paradigmatic example is the right of *ownership* of an object. It basically gives the owner the right to deal with the thing at her/his discretion and to exclude others from exercising any influence whatsoever, and can be transferred to others (Wilhelmi 2023).

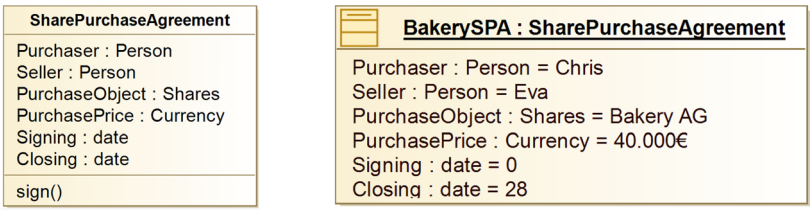
It is also important to consider the possibility of *changes in the legal position*, including the *transfer*. Claims can *arise*, become *due* or *not enforceable*, change, or extinguish as a result of actual events and legal transactions. The ownership of an object can be *transferred* from one legal subject to another, usually by means of a contract.

In the case of an SPA, the object of purchase are the shares of a company (*share deal*) or the assets held by a company (*asset deal*) (Wilhelmi 2024b). The purchaser has a claim against the seller to transfer the ownership of the shares or the assets to her-/himself. Vice versa, the seller has a claim against the purchaser to transfer the purchase price. Both claims are primary claims and aimed at a transfer. In addition, a SPA usually contains *warranty* claims as secondary claims that give the buyer certain rights against the seller if certain warranties are breached.

2.2 Object oriented modeling

We model the legal relationships of the entities in an SPA using an object-oriented modeling approach. We illustrate this modeling using a fictitious SPA in which `Eva` sells the company `Bakery AG` to `Chris`. In an object-oriented modeling approach, an object describes an element of the real world, sometimes also referred to as an instance. Instances of the same type are typically aggregated into classes, where the class then defines the type of the instances that form its membership. The Unified Modeling Language (UML) (Object Management Group 2017) is the prevailing modeling formalism for object oriented systems. For the models that we propose in this paper, *class* and *object* diagrams are the most relevant diagram types of the UML. Class diagrams represent the classes to which objects are aggregated as well as the interrelations among classes. Object diagrams depict the relationships of object instances.

As illustrated in Fig. 2, in the `Pretzel Bakery` example that we consider, the type of contract that we use is that of a `SharePurchaseAgreement`, which



(a) Class `SharePurchaseAgreement` (b) Object `BakerySPA`

Fig. 2 The object `BakerySPA` instantiates the class `SharePurchaseAgreement`

defines a class with that type, see the UML class diagram in Fig. 2aA class can define *attributes* in the first box below the class name. An attribute represents properties of an object and has a name and a type. For example, an SPA will have an attribute `signing` of type `date`, which contains the date when the contract was signed, see Fig. 2a. The optional box beneath the attributes lists the *operations* that the elements of a class can perform. An operation represents a capability of the objects of some class, in the example in Fig. 2a for instance the ability that a contract can be signed, indicated by the operation `sign()`. It sets the attribute `signing` to the value 0. In the later analyses, it is assumed that the contract starts on day 0 and this value will be assigned to a logical variable in the formalization of the SPA.

A UML object diagram illustrates possible values that the attributes of a particular instance of some class can attain. The object `BakerySPA` in Fig. 2b describes an instance of the class `Share Purchase Agreement` and represents the fictitious SPA that we use for illustration purposes in this paper, see Fig. 2b for the various attributes defined in the class `Share Purchase Agreement`, e.g., the names of the `Purchaser` and the `Seller`.

Central to the notion of object-oriented modeling is the idea of modeling at different levels of abstraction, using different views on a system. In this context, an important technique is that of refinement, where a more abstract view of a system is replaced by a more concrete one, for instance one that reveals more detail of the internal structure of the considered system. In particular, refinement can be achieved by replacing an abstract view on the system, for instance given by a single class, by a more concrete view defined by a collection of more concrete classes as well as a description of the relationship among these more concrete classes. A further type of refinement is accomplished in object-oriented modeling by the concept of specialization through inheritance. Here, a more abstract class is specialized into a number of more concrete classes, so that the more concrete classes inherit the attributes and operations of the parent class, while additional attributes and operations can be added to the more concrete class.

UML class diagrams permit the modeling of systems using refinement, specialization and aggregation. Classes are represented by rectangles, and relationships among classes, referred to as associations, are depicted by lines connecting classes. These lines may have text labels denoting the name of the association, the role of the classes as well as the cardinalities of the association at either end of the association. Further triangle- and diamond-shaped labels qualify an association as representing inheritance and aggregation, respectively. In the following we will illustrate these concepts in principle using the class diagram in Fig. 3, the meaning of which will be explained in more detail in Sect. 3. The class diagram in Fig. 3 represents a refined, more concrete view at the concept of a Share Purchase Agreement. Central is the concept of a claim, that we use here to illustrate the above mentioned UML modeling concepts.

The class `Claim` is *specialized* through *inheritance* into either a class `PrimaryClaim`, or a class `Secondary Claim`, as indicated by the association lines that connect these classes and that begin with a triangle on the side of the more general class, pointing to it. Inheritance means that all attributes and operations of the

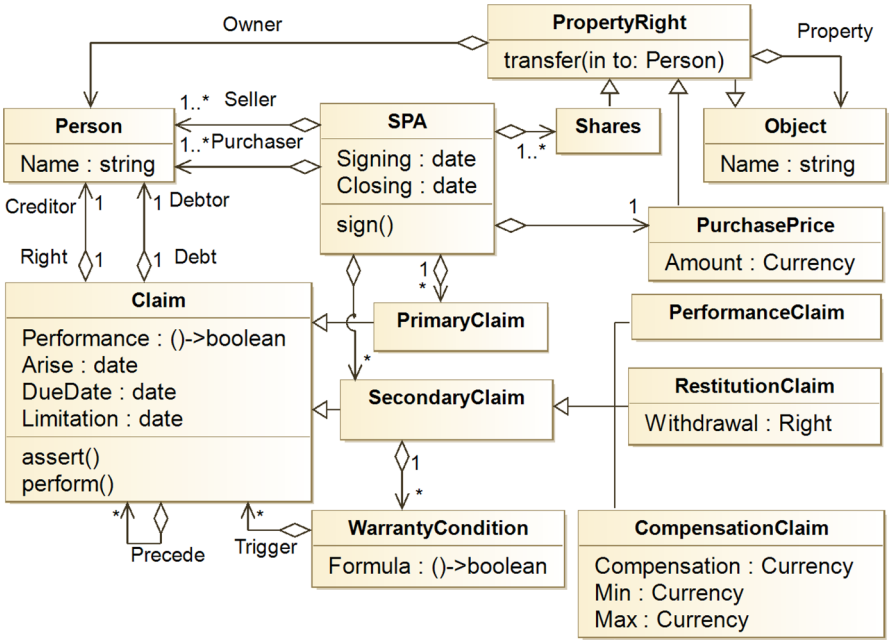


Fig. 3 Class diagram of a share purchase agreement (SPA)

inheriting class are inherited by the specialized class. For instance, the class `PrimaryClaim` will inherit attribute `DueDate` as well as operation `assert` from class `Claim`. Notice that the reverse direction of *specialization* is also referred to as *generalization*, i.e., the classes `PrimaryClaim` and `Secondary Claim` can be generalized to the class `Claim`.

The association type *aggregation* represents a part-whole relationship. As indicated by the aggregation associations between classes `Claim` and `Person`, a claim consists of one reference to a person where the person has the role `Creditor`, and another where the person has the role `Debtor`. In the first case the claim is a `Right` and in the second case a `Debt`. The cardinalities of the associations are indicated by the numbers or ranges at either end of an association. For instance, there is a 1:1 relation between `Claim` and `Person` along the `Right` / `Creditor` aggregation association.

2.3 Decidable fragments of first-order logic

We formalize the logical constraints implied by the clauses of an SPA in decidable fragments of FOL. A fragment of FOL is decidable if a complete and terminating algorithm exists that decides for any formula from this fragment, whether the formula is satisfiable. General FOL is undecidable, which is why we restrict our modeling to decidable fragments of it.

The formalization in this paper uses an FOL fragment which includes the decidable theories of linear real arithmetic, equality, integers and uninterpreted functions (Kroening and Strichman 2016). Efficient SMT solvers (Kroening and Strichman 2016), which automatically decide the satisfiability of formulae from these fragments, are available. In case of satisfiability, they return a satisfying assignment, also referred to as a model, of values to the logical variables. Otherwise, they return the result *unsat* as well as an unsatisfiability core, which contains an explanation for the unsatisfiability. For instance, an SMT solver determines that the formula $(x > 0) \wedge (x + y < 0)$ is satisfiable and may, for example, return the variable assignment $x = 0.9$ and $y = -2.0$ as a satisfying model. When the considered logical constraint system is structured as a partial MaxSMT instance using what is referred to as "soft assertions", some SMT solvers, such as for instance Z3, have the ability to return a maximal set of constraints that are necessary to satisfy an otherwise unsatisfiable model. We will take advantage of this capability when prioritizing a certain type of claim in the analysis.

3 Contract modeling

We propose the use of UML class diagrams in the ontological modeling of an abstract view of an SPA. An instance of a class diagram is then used to represent a concrete legal contract. For legal entities that an expert finds in a concrete legal contract, s/he instantiates classes and adds values and relations given in the contract by assigning attributes. The combination of the objects and their assignments is represented using an object diagram. This information will be inserted in the blocks which are then automatically processed by the *ContractCheck* tool. The set of blocks obtained in this way will be the basis for the logical consistency analyses that we present in later sections. We illustrate this approach by applying it to the modeling of a concrete example SPA regulating the purchase of a pretzel bakery.

3.1 Modeling of an SPA

For modeling purposes, it is necessary to determine the typical provisions in an SPA. These can be found in numerous legal template books. For the present project, we have analysed a large number of templates in a number of form book on German law (von Hoyenberg 2020a, b; Meyer-Sparenberg 2022a, b; Pfisterer 2022; Seibt 2018a; Seibt 2018; Seibt 2018c, d). We have identified the following provisions as typical for an SPA: contracting parties, subject matter of purchase, purchase price provisions, conditions and execution, warranties and indemnities, liability, and final provisions.

Based on this, an example SPA containing these typical provisions, which governs the sale of a pretzel bakery, has been developed. We use this SPA as a running example in this paper. We are mainly concerned with the provisions related to *warranties* and *liability*,[2], supplemented by the indispensable provisions concerning

[2] We set legal terms in *italics* and terms referring to UML diagrams in `teletype` font.

the contracting parties (*purchaser* and *seller*), the *purchase object* and the *purchase price*.

From these provisions, we derive an ontology of the provisions in an SPA that is given as a UML class diagram, presented in Fig. 3. It represents the provisions of an SPA in the form of UML classes. Note that this ontology can be extended to other types of contracts. In this paper, we restrict ourselves to considering SPAs since, compared to other types of contracts, they rely much less on implicit legal facts implied by legal dogmatics, since these are based on the statutory provisions, which are typically abandoned in SPAs.

The essential *claims* in an SPA are the *claims* between the *purchaser* and *seller* for the *shares* in the company and the *purchase price* as well as related *claims* from *warranties*. We illustrate their relationship using a UML Activity Diagram depicted in Fig. 4.

When the *contract* is *signed* (`Signing`), the *claim* for payment of the *purchase price* (`PayClaim`) and the *claim* for transfer of the *shares* (`TransferClaim`) *arise*. In contrast, a *claim* resulting from a warranty (`WarrantyClaim`) only becomes effective if additionally a condition is satisfied.

For each `Claim` one person (`Creditor`) can demand the *performance* and another person (`Debtor`) owes the *performance* as a duty. The *performance* can be a behavior or the production of an outcome (`Performance`). For the `PayClaim`, the `Creditor` is the *seller*, the `Debtor` is the *purchaser* and the `Performance` is the payment of the *purchase price* by the *purchaser* to the *seller*. For the `TransferClaim`, the `Creditor` is the *purchaser*, the `Debtor` is the *seller* and the `Performance` is the transfer of the *shares* by the *seller* to the *purchaser*.

A `WarrantyClaim` additionally requires the breach of the *warranty* as `WarrantyCondition` to become effective. Otherwise, the *creditor* cannot enforce it and its assessment ends, whereas the assessment of the other `Claims` is independent of this. `WarrantyClaims` can be distinguished into three types, depending on the content of their *performance* (`PerformanceContent`): the *performance* of the content of the *warranty* (`PerformanceClaim`), the *restitution* of the *purchase price* (`RestitutionClaim`) and the *compensation* of the damages of the *purchaser* (`CompensationClaim`). A `CompensationClaim` is often additionally limited in the way that the damage must reach a minimum value (`Min`), otherwise the *creditor* cannot enforce it and its assessment ends, whereas the assessment of the other `Claims` is independent of this. For the the three `WarrantyClaims`, the `creditor` is the *purchaser* and the `debtor` is the *seller*.

Each of these `Claims` independently not only needs to have *arisen*, but also needs to have not *extinguished*, be *due* and not yet fall under the *limitation*. If the owed `Performance` is rendered, the `Claim` extinguishes and the `creditor` cannot assert (and enforce) it. If the `Claim` is not yet `Due`, which here means before the `Closing`, the `creditor` cannot yet enforce the `Claim`, but the `debtor` can still satisfy it in rendering the `performance`. If the `Claim` is due, the `creditor` can assert and enforce it and the `debtor` has to satisfy it up to the `LimitationDate`. From the `LimitationDate`, the `creditor` can enforce the `Claim` only, if the `debtor` does not raise the *defense of the statute of limitations* (`LimitationDefense`), otherwise the `creditor` cannot enforce the `Claim`.

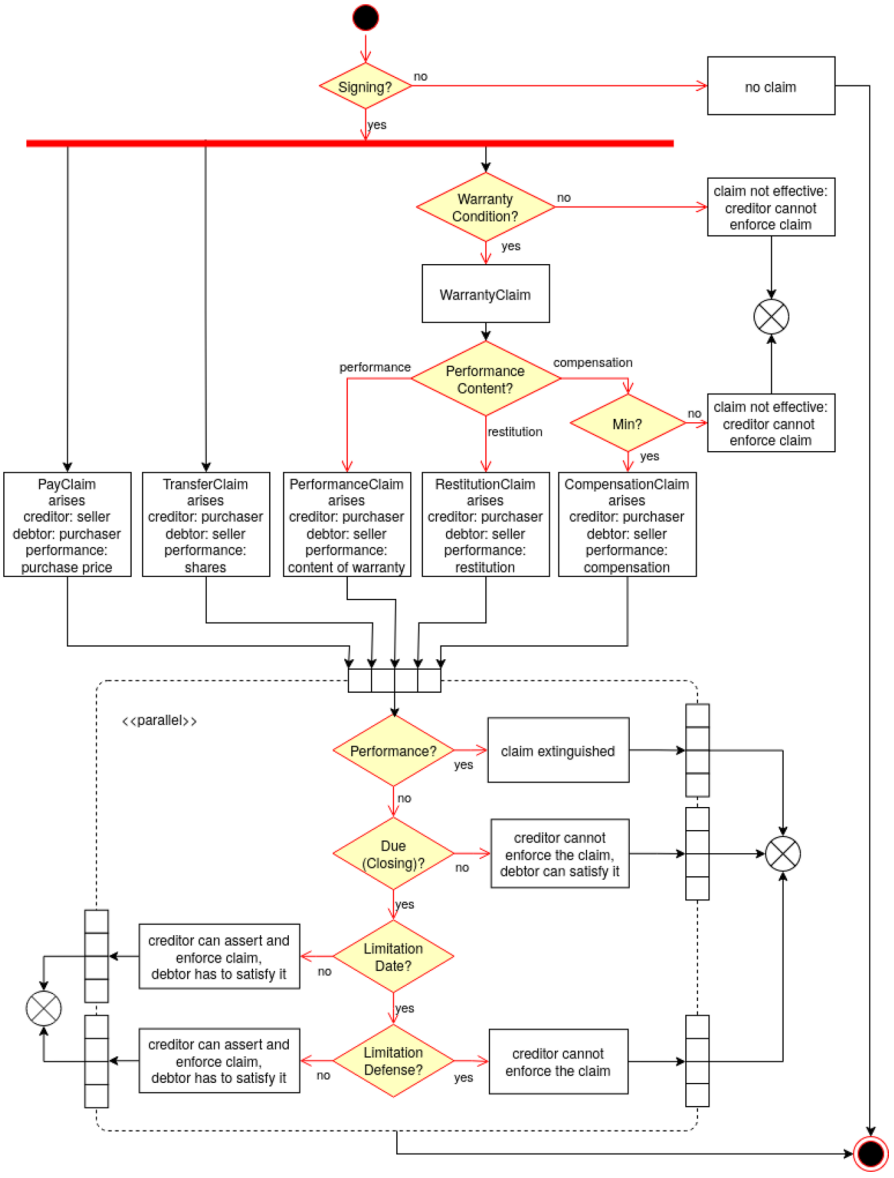


Fig. 4 Claim arise and fulfillment in the Bakery contract

The respective run ends when all `Claims` have been examined. The *contract* ends when all `Claims` have extinguished or are no longer enforceable.

In the UML class diagram, depicted in Fig. 3, each `SPA` has at least one `Person` as `Seller` and another `Person` as `Buyer`. For each `Claim`, either the `Purchaser` or the `Seller` is the `Debtor`, who owes the `Performance` as `Debt`, and the other one is the `Creditor`, who has the `Right` to demand the

Performance in asserting the Claim. For each Claim, either the Purchaser or the Seller is the Debtor and the other one is the Creditor. The Creditor has the Right to demand, that the Debtor pays his Debt in rendering the Performance. The exercise of this Right is described as the operation assert. The promised Performance is described as an attribute in the Claim. The date from which the Claim is due, i.e. the date from which the Creditor can assert it, is described with the attribute DueDate. The date from which the Debtor is entitled to refuse Performance of the Claim in raising the *defense of the statute of limitations* is described with the attribute LimitationDate. A *claim* is extinguished if the *performance* owed is rendered.

We distinguish between PrimaryClaims and SecondaryClaims. PrimaryClaims contain the Performances that are typical for the contract. They are performed if there is no disturbance of the contract. The *claims* to transfer the *shares* and the *purchase price* are PrimaryClaims. To cover their Performance, we need to model the Shares and PurchasePrice and their generalization property right, because the Debtor performs the Claim, when s/he is the Owner of the PropertyRight in the Shares or the PurchasePrice and transfers this right to the Creditor. The PrimaryClaims have to be performed from the outset by the DueDate, sometimes under the condition of the Performance of another PrimaryClaim. In contrast, SecondaryClaims arise only if the contract is disturbed. They are subject to a specific condition, which we refer to as a WarrantyCondition. This condition may consist of the breach of another *claim*, often a PrimaryClaim. It may also consist of the breach of separately enumerated circumstances as a condition independent of the breach of another *claim*, which is typical of SPAs. We model these conditions with the attribute Formula. SecondaryClaims for which the WarrantyCondition entails the breach of another Claim, are *consequence claims*. In this case, the link between the WarrantyCondition and the other Claim is represented by an association named Trigger. SecondaryClaims for which the WarrantyCondition only entails separately enumerated circumstances, are called *independent warranty claims*. They are *independent claims* if they do not refer to other claims.

The most important *secondary claims* in an SPA are *warranty claims* (Wilhelmi 2024c). A WarrantyClaim relates to an unknown risk arising from a breach of a *primary claim*, or from listed circumstances that are not expected but are considered probable enough to require regulation. A *warranty* consists of the *warranty content* as a prerequisite and one or more *warranty claims* as a consequence of a breach of the *warranty*. The *warranty content* includes the existence or non-existence of certain circumstances, which usually concern the *purchase object* and, in particular, its properties. It contains the WarrantyCondition for the warranty claims. They are usually *claims* of the *purchaser* for monetary compensation for the damage caused by the breach of the WarrantyCondition, irrespective of fault (CompensationClaim). It is also possible to agree on a *claim* for subsequent performance (PerformanceClaim) or a right of Withdrawal which, in the case of a notice of withdrawal (noticeWithdraw), terminates the contract and may give

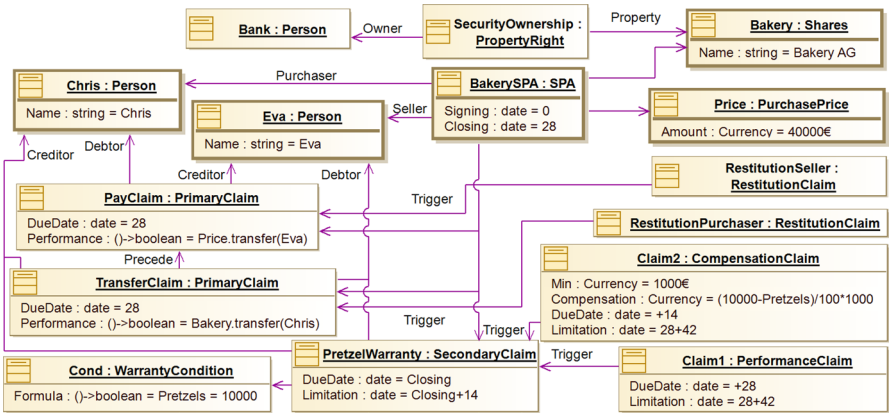


Fig. 5 Object diagram of Bakery SPA (with only partial depiction of the associations to avoid clutter)

the purchaser a *claim* for restitution of the *purchase price* and the seller a *claim* for restitution of the *shares* (`RestitutionClaims`). A `CompensationClaim` is often limited in the way that it only arises, if the *Damage* reaches a minimum value `Min`, and the `Compensation` is capped at a maximum value `Max`. The limitation may apply only to individual *claims* or to all *claims* under the contract. If the *purchaser* has a *claim* for compensation, this may be set off against the purchase price claim, so that the total amount of money to be paid is reduced accordingly.

The due date of a consequence claim is usually relative to the assertion date of a warranty. The contract model supports absolute and relative time relations. Relative time differences start with the plus sign + and are relative towards the occurring or asserting date of the claim to which the `Trigger` refers. Examples are given in Fig. 5. The `PerformanceClaim` has to be performed within +28 days after the `PretzelWarranty` was asserted. The limitation 28 + 42 of the claim is given as an absolute value.

A claim can also have an association `Precede` to another claim. A *claim* does not become due without its *preceding claim* being performed beforehand. For instance, the payment of the *purchase price* may be expected to occur before the transfer of the *shares*. Otherwise, the seller can defense the share transfer because of the lack of payment. In this case, the `Claim` for the `PurchasePrice` is a *preceding claim* that has the *performance* of the `Claim` for the `Shares` as a condition. In the diagram in Fig. 3, the link between the *claim* and the *preceding claim* on whose occurrence it defenses is represented by a self-association named `Precede`.

Example 1 (Bakery SPA) *As a running example, we use an SPA for the fictitious sale of a pretzel bakery from the seller* `Eva` *to the purchaser* `Chris`.[3] *This SPA is a concrete instance of the SPA type depicted in the class diagram in Fig. 3. This concrete instance is shown in the UML class diagram presented in Fig. 5. In the bakery*

[3] The text of the Bakery SPA is given in the "Appendix".

SPA, *Chris agrees to pay the purchase price of* €40, 000 (`PayClaim`), *and Eva agrees to give Chris ownership of the bakery by transferring* `Shares` *of the* `Bakery`, *which constitutes the* `TransferClaim`. *Both claims are due on the agreed closing date of* 28 *days after signing*. *If the pretzel bakery is not transferred or the purchase price is not paid at closing*, *the other party may withdraw from the purchase agreement*. *The conditions about arise and fulfillment of claim in the Bakery SPA is depicted in the activity diagram in Fig*. 4.

In addition to the primary claims, *Eva provides a warranty in a* `WarrantyClaim` *named* `PretzelWarranty` *which states that the bakery can bake at least* 10, 000 *pretzels a day*. *In the example*, *if the bakery cannot bake at least* 10, 000 *pretzels a day*, *then the warranty is breached and Chris must assert this breach within the* `DueDate` *of* +14 *days after closing*. *In the event of an assertion*, *Eva has to make good within* +28 *days because of the* `PerformanceClaim` `Claim1`, *otherwise she has to pay within* +14 *days a compensation of* €1, 000 *per* 100 *of pretzels that cannot be baked*, *due to the* `CompensationClaim` `Claim2`, *which entails a minimal compensation of* €1000. *Any claim under the warranty has a* `Limitation` *of* 42 *days from closing*. *Other crucial facts are represented in the bakery SPA*. *Because of Eva's debts*, *a local bank* `Bank` *has a security interest in the shares of the bakery* `SecurityOwnership`. *Under German law*, *this means that the bank is the* (*security*) *owner of the bakery*.

Manual inspection reveals the following inconsistencies in the SPA: *Eva's primary claim according to the SPA is to transfer the bakery to Chris*. *To satisfy her claim*, *she must be the owner*. *Note*, *however*, *that the Bank is the owner of the bakery by virtue of the transfer of ownership by way of security*. *Eva cannot fulfill her claim because she is no longer the owner of the bakery*, *which we consider an inconsistency*. *While there are legal ways for dealing with this situation*, *the objective of our consistency analysis will be to point out such stipulations that require further attention*. *Another inconsistency is due to the timing of the claims*. *The* `PretzelWarranty` *has a* `Limitation` *of* 42 *days but if an assertion occurs within* 14 *days after closing*, *the consequence* `Claim1` *takes up to* 28 *days and* `Claim2` *another* 14 *days*. *This implies that* `Claim2` *can take up to* 56 *days after closing*, *contrary to the* `Limitation` *of* 42 *days*.

4 Formalization

We present the logical encoding of the objects of an SPA in decidable fragments of FOL, as this turns out to be ideally suited to accommodate consistency checking for a large variety of possible legal scenarios, including ones that are not known at the time of signing. The versatility of available decidable first-order theories allows us to capture a large range of domain constraints in SPAs.

We then demonstrate the applicability of the proposed encoding using the pretzel bakery SPA case study that we introduced above. The aim of this encoding is to enable a model-based satisfiability analysis to determine logical consistency of the provisions of an SPA. A satisfying model for the logical encoding represents a possible execution of the considered SPA. An unsatisfiable encoding, on the other hand, hints at either inconsistent claims or even a non-executable contract. We intend the consistency analyses in Sect. 5 to be performed primarily before signing, when the legal facts are only partially known.

4.1 Logical formalization of contract entities

The quintessential concept of a contract is that of mutual claims, i.e., the right of a legal subject to demand an act or omission from another legal subject. An SPA consists of a set of claims with which an execution of the SPA must comply. Whether a claim is performed in an execution depends on the objective situation and the behavior of the contracting parties, which we call *legal facts*. We use variables over various domains to model these facts, such as integers to model purchase prices and dates, and first order predicates to model property relations. We restrict their possible values according to the constraints specified in the SPA. These facts are then used to specify pre- and postconditions for the execution of claims as they can be extracted from the SPA.

We define the *execution* of a contract as the performance of a sequence of claims. A particular execution of a claim is determined by a combination of claims and the satisfaction of their preconditions, which includes factual and legal information given by the provisions of the contract, such that for each *primary claim* or *independent claim*, the claim itself or one of its associated *consequence claims* is performed. The pre- and postconditions for the execution of the claims define a weakest precondition semantics of the contract stipulations in the sense that the logical formulae that use in the formalization capture a precondition for the executability of a claim, for instance that the due date of a particular claim has been reached, and a postcondition that expresses that this claim has now been executed. Note that not every claim in an SPA must be performed in a particular given execution.

The types of inconsistencies in claims and among claims encompass the following:

- primary and secondary claims may be performed outside the time interval between their due dates and the limitation date specified in the contract,
- a secondary claim is not performed in this time interval in spite of the fact that the corresponding primary claim is not performed,
- the compensation performed by executing a warranty claim is not within the limits specified in the contract,
- numerical values, such as the purchase price or a numerical performance characteristic, are not within specified limits, and

- an ownership relation is violated, e.g., by selling an item that the seller is not the owner of.

Notice that this list is not claiming to exhaustively cover all possible inconsistencies in contract texts. Rather, it covers a large portion of inconsistencies that are encountered in legal practice.

First, in Sect. 4.1.1 we formalize the legal facts, then in Sect. 4.1.2 we provide abstract formalizations of the types of claims that we analyze, and finally in Sect. 4.1.3 we compose the formalization of individual claims in such a way that a satisfying assignment of the composed formula corresponds to a possible SPA execution.

4.1.1 Formalization of legal facts

An SPA according to the class diagram in Fig. 3 contains a set $\mathcal{P}$ of persons and a distinct set $\mathcal{O}$ of objects. In the formal model that we build we assume that the persons p and the objects o have unique identifiers.

Dates in an SPA are usually given by calendar dates. Calendar dates are complicated to process due to the many rules that govern them, such as the treatment of leap years. In the formalization, we simplify the processing and represent dates using integer variables. An execution of an SPA begins on date $d_S = 0$ with the signature of all contract parties, referred to as the *signing*. The *closing* is performed on date d_C when the transfer of the business ownership is performed. For each claim c in an SPA, we define a date d_c which represents its date of performance.

An ownership can only be transferred from the current owner to a new owner if the ownership rights lie with the seller. As depicted in Fig. 3, Property relations relating an object to an owner can be represented by associating the class `PropertyRight` using an association `Owner` with a `Person` p, and using an association `Property` with an `Object` o. We define a set $\mathcal{PR}$, consisting of a tuple (p, o) for every instance of `PropertyRight`, in the object diagram representing the contract instance that is being analyzed, c.f. Fig. 5. In our analysis model, we formally capture the property rights using a first-order uninterpreted function predicate $owner(Object) : Person$:

$$\phi_{owner} = \bigwedge_{(p,o)\in\mathcal{PR}} owner(o) = p. \tag{1}$$

The first-order formula ϕ_{owner} then represents the property relations stated in the SPA.

4.1.2 Formalization of claims

Having formalized the legal facts stated in an SPA we are now ready to formalize the claims that it contains. Notice that we refer to classes, object instances, operations, association names and association roles as they are defined in Figs. 3 and 5.

We define a set $\mathcal{C}$ to consist of the claims stated in an SPA. The set $\mathcal{C}_I$ contains the *primary claims* and the *independent warranty claims* of the considered SPA. For every *claim* c, the set $\mathcal{C}_c$ contains every *secondary claim* s that is connected via the association `Trigger` to c. In the following, we refer to a claim in $\mathcal{C}_c$ as a *consequence claim* of c.

Each claim c in the SPA contains a `Performance` l_c, which specifies an action that must be executed for the claim to be performed. The action to which l_c refers is either represented by a logical formula or by the name of an operation in Fig. 3. In the latter case, l_c is replaced by the logical formalization of the pre- and postconditions of the corresponding operation. For example, the precondition for a property transfer p of an object o from the debtor d to a creditor in terms of the property rights is represented by the constraint $l_c^p \equiv owner(o) = d$.

For every primary or secondary claim c there exists a performance date d_c. The value of d_c is either -1, which we use to express the fact that the claim has not yet been performed, or in the interval [`DueDate`, `Limitation`]. If `DueDate` is undefined in the considered SPA and, consequently, in the corresponding object diagram, then its default value is the value of date `Arise` of the claim. In case `DueDate` is a positive number, then we replace it with `Arise` + `DueDate`. In conclusion, we formalize a claim c using the logical constraint

$$\phi_c \equiv (d_c = -1) \vee ((c.\texttt{DueDate} \leq d_c \leq c.\texttt{Limitation}) \wedge l_c). \tag{2}$$

In order to relate this to the pre-/postcondition semantics that we allude to above, notice that the subformula $c.\texttt{DueDate} \leq d_c \leq c.\texttt{Limitation}$ represents a precondition for the executability of c, and the subformula l_c expresses the postcondition that c has now been executed.

A `WarrantyClaim` w is a specialization of a *secondary claim* in the sense that it is an independent claim. For this reason, in our formalization, the constraint $d_w = -1$ encodes that w is met, and otherwise it is breached on a date $d_w \geq 0$. A warranty w is formalized by

$$\phi_w \equiv (d_w = -1 \Rightarrow l_w) \vee (w.\texttt{DueDate} \leq d_w \leq w.\texttt{Limitation}). \tag{3}$$

If a *primary claim* or an *independent claim* is not performed, the `Debtor` is obliged to perform a consequence claim s that is associated with the *claim*. A *primary claim* is breached when it is not performed ($d_c = -1$), while a warranty is breached on the date d_w of the notification that the `WarrantyCondition` is not met. In order to formally capture the consequence claims that belong to a claim, we introduce a fresh integer variable d'_c with value -1 when the associated *claim* is performed. The value of d'_c is $c.\texttt{DueDate}$ for a breached *primary claim* and the value of d_c for a breached warranty. In case s is a performance claim, its formalization with a performance date d_s and a `Performance` l_c is defined as $\phi_s \equiv (d'_c < d_s \leq s.\texttt{Limitation}) \wedge l_c$. In case s is a restitution claim, the SPA is withdrawn, formally $\phi_s \equiv d'_c < d_s \leq s.\texttt{Limitation}$. In case s is a compensation claim, the debtor pays a positive compensation l_s to the creditor. The value of l_s is the `Compensation` that specifies the amount of the compensation payment. As expressed

by Formula 4, l_s is constrained to be in a range between a minimum amount `Min` and a maximum amount `Max`.

$$\phi_{Compensation} \equiv l_s = \begin{cases} 0, & \textit{if Compensation} < \texttt{Min} \\ \texttt{Max} & \textit{if Compensation} > \texttt{Max} \\ \textit{Compensation}, & \textit{otherwise} \end{cases} \tag{4}$$

In every of the above cases, the compensation is performed. In conclusion, the compensation claim is formally a constraint

$$\begin{aligned} \phi_s \equiv \phi_{Compensation} \wedge ((d_s = -1 \Rightarrow \textit{Compensation} = 0) \vee \\ (d'_c < d_s \leq s.\texttt{Limitation})). \end{aligned} \tag{5}$$

4.1.3 Execution of an SPA

We are now prepared to present the logical formalization of the executions permitted by an SPA by a formula ϕ_{SPA}, given in Eq. 6. It captures the property rights and the claims included as an SPA by *conjoining* their above given formalizations. It also ensures that in an execution of an SPA, for every *primary* and every *independent claim*, either the claim or one of its consequence claims will be performed.

$$\phi_{SPA} \equiv \phi_{owner} \wedge \bigwedge_{c \in \mathcal{C}} \phi_c \wedge \bigwedge_{c \in \mathcal{C}_I} (d_c \geq 0 \vee \bigvee_{\forall s \in \mathcal{C}_c} d_s \geq 0). \tag{6}$$

An SMT solver, such as Z3, will produce a satisfiable model for ϕ_{SPA} in case such a satisfiable model and, consequently, an execution of the SPA exist. This model then represents an execution of the SPA that is consistent with all constraints specified in the SPA and formally represented by ϕ_{SPA}.

During the execution of a contract, the contracting parties often have a choice as to whether to fulfill a *primary claim* or an *independent claim*, or to breach that *primary claim* or *independent claim* and be subject to a *secondary claim* that results from the breach. This second option may put a contracting party into an advantageous position ("efficient breach"). It is not the objective of our work to determine advantageous courses of action for the contracting parties. Nonetheless, in the analysis it might be worthwhile to determine whether an execution of the contract involving just *primary claims* is at all possible, in order to obtain a deeper understanding of the functioning of the considered contract. To support this idea, we propose a logical encoding that permits the prioritization of *primary claims* over *secondary claims* during analysis. We introduce a set ϕ_{soft} of logical assertions that is encoded in Z3 using what is referred to as "soft-asserts", c.f. Eq. 7.

$$\phi_{soft} \equiv \bigwedge_{c \in \mathcal{C}_I} d_c \geq 0 \ \wedge \bigwedge_{s \in \mathcal{C}_c} d_s = -1 \tag{7}$$

The SMT solver Z3 computes an optimal solution for the partially satisfiable Max-SMT problem $\phi_{SPA} \wedge \phi_{soft}$. The computed solution is optimal in the sense that the smallest number of assertions in ϕ_{soft}, each corresponding to a breach of a primary or independent claim and the execution of a secondary claim, is made satisfied, which means that the corresponding secondary claim will be executed. As a consequence, Z3 returns a satisfying model representing an execution of the SPA that satisfies as few consequence and independent claims as possible.

4.2 Formalization of the Bakery SPA

We now illustrate the application of the formalization described above to the Pretzel Bakery SPA case study.

4.2.1 Legal facts in the Bakery SPA

The bakery SPA B describes a set of persons $\mathcal{P}^B = \{\texttt{Eva}, \texttt{Chris}, \texttt{Bank}\}$ and an object set $\mathcal{O}^B = \{\texttt{Bakery}, \texttt{Price}\}$ which includes the shares of the `Bakery` and the purchase price `Price`. The SPA also states that the ownership of `Bakery` is transferred by way of security to `Bank`, encoded as

$$\phi^B_{owner} \equiv owner(Bakery) = Bank. \tag{8}$$

4.2.2 Claims in the Bakery SPA

Eva is obliged by the `TransferClaim` to transfer the shares of the bakery on a day d_u. She has to perform the `TransferClaim` on the day of `Closing` (28). In case she misses that date, `Chris` has a claim on her delivery of the shares. Eva can only perform the transfer if she is the owner of `Bakery`. These facts are captured using the constraint

$$\phi_{TransferClaim} \equiv (-1 = d_u) \vee ((28 \leq d_u) \wedge (owner(Bakery) = Eva)). \tag{9}$$

Furthermore, Chris is obliged by the `PayClaim` to pay the amount of `Price` to Eva. The condition for the transfer is formally captured by the constraint $owner(Price) = Chris$. The transfer is performed on a day d_z and is due on day 28. The formalization of the `PayClaim` is then given as

$$\phi_{PayClaim} \equiv (-1 = d_z) \vee ((28 \leq d_z) \wedge owner(PurchasePrice) = Chris)). \tag{10}$$

If one of these two *primary claims* is not performed, then the related `RestitutionClaim` (shorthand: Res.) allows the respective creditor on a date ≥ 0 to withdraw from the SPA, formally captured by the formulas

$$\phi_{Res.Purchaser} \equiv d_{Res.Purchaser} = -1 \vee PayClaim.DueDate < d_{Res.Purchaser} \tag{11}$$

$$\text{and } \phi_{Res.Seller} \equiv d_{Res.Seller} = -1 \vee TransferClaim.DueDate < d_{Res.Purchaser}. \quad (12)$$

If the *warranty claim* `PretzelWarranty` is breached, then Chris notifies Eva of this breach on a day $d_g \geq 0$. Notice that $d_g = -1$ is used to encode that there is no indication of a non-performance and, therefore, the warranty `Condition`, which stipulates that $Pretzels = 10,000$, is met. The formalization for the `PretzelWarranty` is then given as

$$\phi_{PretzelWarranty} \equiv (d_g = -1 \wedge Pretzels = 10,000) \vee (28 \leq d_g \leq 28 + 14). \quad (13)$$

In the bakery SPA, the warranty has the consequence `Claim1` of type `PerformanceClaim`, which implies that the seller must perform the pretzel guarantee on a date d_n in case the warranty is not met. This is encoded using the following formula:

$$\phi_{Claim1} \equiv (d_n = -1) \vee (d_g < d_n \leq d_g + 28 \wedge Pretzels = 10,000). \quad (14)$$

On the other hand, following the idea of an efficient breach it may be more advantageous for the debtor to pay a compensation l_s on a date d_s according to the attribute `Compensation` of the object `Claim2` of type `CompensationClaim`. The value of l_s is constrained by the formula $\phi^s_{Compensation}$. If no compensation occurs ($d_s = -1$), then l_s is 0. The formalization of the logical constraint encoding `Claim2` is

$$\phi_{Claim2} \equiv \phi^s_{Compensation} \wedge ((d_s = -1 \wedge l_s = 0) \ \vee (d_g < d_s \leq d_g + 28 + 14)). \quad (15)$$

In order to concretize $\phi^s_{Compensation}$, consider that the amount of compensation l_s is either 0, or lies in the range of values between a minimum value `Min` and a maximum value `Max`. In order for a compensation to be paid, l_s must exceed the value of €1, 000, while an upper limit `Max` is not specified. l_s is calculated according to the constraint $\phi^s_{Compensation}$ defined as follows:

$$\phi^s_{Compensation} \equiv l_s = \begin{cases} 0, \textit{if } (10,000 - Pretzels/100) * 1,000 \leq 1,000, \textit{and} \\ (10,000 - Pretzels/100) * 1,000, \textit{otherwise.} \end{cases} \quad (16)$$

4.2.3 Execution of the Bakery SPA

The overall encoding of the claims in the bakery SPA formalized in Eq. 6 is given as

$$\phi^B_{SPA} \equiv \phi^B_{owner} \wedge (\phi_{TransferClaim} \vee \phi_{Res.Purchaser}) \wedge (\phi_{PayClaim} \vee \phi_{Res.Seller}) \wedge (\phi_{PretzelWarranty} \vee \phi_{Claim1} \vee \phi_{Claim2}). \quad (17)$$

The formula 17 encodes the freedom to perform a claim or one of its consequences by a *disjunction* of the individual claim constraints. In order to express that all claims or their consequence claims need to be performed, we *conjoin* the

subformulae related to the individual claims and the legal facts to form the constraint ϕ^B_{SPA} that represents the encoding of all claims in the SPA. Notice that an execution of the bakery SPA does not necessarily need to perform every *primary claim*, but can also perform the associated *consequence claim* for some or all of the *primary claims*. However, a satisfying model of ϕ^B_{SPA} should preferably represent the performance of a sequence of *primary* or *independent claims*. This is formalized by the following formulas which are characterized as "soft-asserts" in the smtlib encoding handed over to the SMT solver:

$$\begin{aligned}\phi^B_{soft} \equiv d_u \geq 0 \wedge d_z \geq 0 \wedge d_{Res.Purchaser} = -1 \wedge \\ d_{Res.Seller} = -1 \wedge d_g = -1 \ \wedge \ d_n = -1.\end{aligned} \tag{18}$$

The bakery SPA is formalized by the constraint $\phi^B_{SPA} \wedge \phi^B_{soft}$. Each satisfying model of this constraint represents a possible execution of the bakery SPA.

Number of contract executions The steps of a contract execution can be distinguished into the performance of claims and into the assignment of variable values. Examples of the latter are the number of possible choices of prices and the days on which the claims are performed. For instance, each choice of a price leads to a distinct execution. Since there are infinitely many possible prices to choose from, there are infinitely many executions. Notice that in general due to the use of variables of type integer or real in the formalization, the number of resulting distinct contract executions is (countably) infinite.

We now argue why, nonetheless, only a finite number of different sequences of claim executions need to be considered. In a contract execution, every *claim* is either performed or not. The claims are related by a *Trigger*. If a *claim* is not performed, another *claim* is triggered and arises. The *claims* of the bakery SPA trigger one another as depicted in the activity diagram in Fig. 6. The claims that are related by a trigger *Trigger* are in the same trigger set. For the Bakery example, Fig. 6 shows 3 trigger sets with 2, 2 and 3 elements. Out of each trigger set, exactly one claim needs to be performed in an execution. This leads to 12 different combinations of claim executions that can be performed in a contract execution.

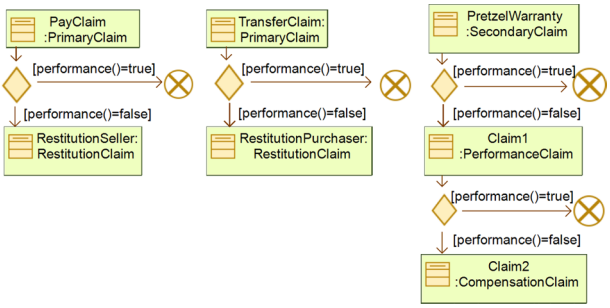


Fig. 6 Claims that are related by a *Trigger* in the Bakery contract

5 Contract analyses

In this section we discuss the syntactic and semantic consistency analyses that we propose, based on the above presented formalization of an SPA.

5.1 Syntactic analyses

Our paper focuses on semantic analyses, based on our logical formalization. However, in addition to these semantic analyses, our analysis framework allows for the analysis of syntactic properties of an SPA. In particular, we can easily perform syntactic completeness checking, since the ontological modeling allows us to infer some syntactic correctness and completeness criteria, e.g. that an SPA lacks essential legal elements. For example, an SPA is not legally valid if one of the essential elements (*essentialia negotii*) is missing. According to the German civil code (BGB, 2024,§433), the contract of sale must contain the parties to the contract, the subject matter of the purchase and the purchase price. In an SPA, every *primary claim* also needs at least one consequential claim and every claim needs a `DueDate` by which it is to be performed. Syntactic completeness checks can easily be performed when parsing the block model of a concrete SPA instance, based on the Class diagram in Fig. 3. However, in the sequel we will focus on presenting the semantic consistency analysis of SPAs.

5.2 Semantic analyses

Finding a dynamic inconsistency that occurs during SPA contract execution requires reasoning about the dynamic semantics of executing an SPA. As discussed above, an SPA describes both legal facts and claims agreed by the parties. For example, Eva agrees to transfer the shares of the bakery in the considered pretzel bakery SPA. In order to transfer ownership of the shares, she must be the owner of the bakery, and hence, the shares. This contradicts the fact that the bank owns the shares in the bakery. Interestingly, in spite of this semantic inconsistency the SPA can be executed or fulfilled because Chris can withdraw from the contract. Nonetheless, an SPA which leaves Chris no option but to withdraw is neither meaningful nor desirable and should therefore be tagged with a "red flag" as a result of the analysis. This illustrates that for an SPA it is essential that a) each claim can be fulfilled individually, and b) that at least one meaningful execution of the SPA exists. Consequently, we propose two types of semantic consistency analyses for an SPA:

Analysis I: Can each claim be performed?

Analysis II: Does there exist an execution of the SPA?

5.3 Formalization of the dynamic consistency analyses

We now encode analyses I and II for a given SPA using the formalized constraints presented in Sect. 4. If an encoded analysis is satisfiable, then an SMT solver, such as Z3, returns a model given by a satisfying variable assignment. The returned model represents either individual claims or entire contract executions that are consistent and can be executed. If it is unsatisfiable, the SMT solver can return an unsatisfiability core, which is a minimal subset of conditions that contradicts satisfiability. This subset indicates which claims in the SPA contradict each other and, hence, cause an inconsistency in the SPA. The respective claims need to be changed in the SPA in order to obtain a consistent contract.

Remember, a *primary claim c* is performed on date d_c and a *warranty w* is asserted on day d_w. In the following, in order to simplify the presentation we abuse notation and ignore that warranties and claims behave differently. In the formulae that follow, $d_c >= 0$ needs to be substituted by $d_w == -1$ for every warranty w.

Analysis I (Claim consistency) is encoded using the constraints Φ_c and Φ_s for every claim in $\mathcal{C}$. We presented the formalization for a claim c by a constraint ϕ_c, which has to be performed on a date d_c in Sect. 4.1.2. For every claim, in order for it to be fulfillable, the property relation represented by ϕ_{owner} has to hold. A *primary claim c* or an *independent claim* $c \in \mathcal{C}_I$ can be performed when the following constraint is satisfiable:

$$\Phi_c \equiv \phi_{owner} \wedge \phi_c \wedge d_c \geq 0. \tag{19}$$

A consequence claim $s \in \mathcal{C}(c)$ in an SPA usually needs to be performed if, at the same time, the claim c that is the `Trigger` of s is not performed:

$$\Phi_s \equiv \phi_{owner} \wedge (d_c = -1) \wedge \phi_c \wedge \phi_s \wedge d_s \geq 0. \tag{20}$$

If every constraint Φ_c and Φ_s is independently satisfiable, then every claim of the considered formalized SPA can be performed.

Analysis II (Contract Executability) checks whether an execution of the SPA exists by assessing the satisfiability of

$$\Phi_{SPA} \equiv \phi_{SPA} \wedge \phi_{soft}. \tag{21}$$

In addition, Analysis II recognizes whether all *primary claims* and *independent claims* can be performed in the same execution, which means that no *secondary claim* in the contract needs to be executed. If that is the case, then every constraint in ϕ_{soft} is satisfied.

5.4 Dynamic consistency analyses of the Bakery SPA

We now apply the dynamic analyses to the pretzel bakery SPA. The object diagram in Fig. 5 represents the information relevant for the analysis, such as the values of dates, the amounts of money and the names of the seller and the purchaser. After reading and parsing collection of blocks representing this contract, *ContractCheck* generates the following analyses and checks them for satisfiability by invoking the SMT solver Z3.

Analysis I (Claim Consistency) checks whether the individual *primary claims* and *independent claims* are fulfillable. The *TransferClaim* is formalized in the Eq. 9 and the corresponding Analysis I is

$$\begin{aligned}\Phi_{TransferClaim} &\equiv \phi^{B}_{owner} \wedge \phi_{TransferClaim} \wedge\ d_u \geq 0 \\ &\equiv owner(Bakery) = Bank \ \wedge (d_u = -1 \vee \\ &\quad (28 \leq d_u \ \wedge owner(Bakery) = Eva)) \ \wedge\ d_u \geq 0.\end{aligned} \tag{22}$$

owner is a function, therefore the two constraints $owner(Bakery) = Bank$ and $owner(Bakery) = Eva$ are functionally inconsistent. As a consequence, the claim cannot be *performed*, which indicates that the contract contains a logical inconsistency. The SMT solver returns these two inconsistent constraints as an unsatisfiability core.

Analysis I for the *PayClaim* (Eq. 10) is performed by determining satisfiability of the following constraint:

$$\begin{aligned}\Phi_{PayClaim} \equiv owner(Bakery) = Bank \ \wedge (d_z = -1 \vee \\ (28 \leq d_z \wedge owner(Price) = Chris)) \wedge d_z \geq 0.\end{aligned} \tag{23}$$

The SPA states that the closing should be performed on day $d_z = 28$. A possible model that an SMT solver computes which satisfies this constraint contains the assignments $d_z = 28$ and $owner(Price) = Chris$ which indicates that Chris pays the purchase price on day 28 after signing. We can conclude that the `PayClaim` can be *performed*.

Analysis I for the *PretzelWarranty* (Eq. 13) is performed by determining satisfiability of the following constraint:

$$\begin{aligned}\Phi_{PretzelWarranty} \equiv owner(Bakery) = Bank \ \wedge\ (28 \leq d_g \leq 28 + 14 \vee \\ (d_g = -1 \wedge pretzels = 10,000)) \wedge d_g = -1.\end{aligned} \tag{24}$$

$\Phi_{PretzelWarranty}$ is satisfiable for $d_g = -1$ and $pretzels = 10,000$. This means that the pretzel bakery meets the warranty requirement and the warranty claim is performed.

The Analysis II (Contract Executability) for the pretzel bakery example is encoded in the constraint $\Phi^{B}_{SPA} \equiv \phi^{B}_{SPA} \wedge \phi^{B}_{soft}$, following Eq. 21. It can be used to check whether an execution of the bakery SPA exists. The SMT solver that we use in our automated analysis searches for a satisfiable assignment and computes, for instance, a model with the following date assignments:

$$d_u = -1, d_z = 28, d_g = -1, d_n = -1, d_s = -1,$$
$$d_{ResitutionSeller} = -1, d_{ResitutionPurchaser} = 29. \tag{25}$$

This outcome implies that the bakery SPA can be performed, even though it is inconsistent since the *primary claim* `TransferClaim` cannot be performed. It exemplifies that the inconsistency of an SPA does not imply that it cannot be performed. Furthermore, as the SMT solver confirms, if the bakery were not assigned by way of security, but instead *owner*(`Bakery`) = *Eva* held, the pretzel bakery SPA would be consistent.

5.5 Further dynamic analyses

We now present some further analyses that address single claims. They differ from Analysis I in the sense that they are not a prerequisite for the Analysis II.

Unsatisfiability of single claims A claim may or may not be satisfied during the execution of a contract. The model of a contract must support both cases. If a claim is not fulfilled, further claims arise as a consequence for the non-fulfillment. We propose a further analysis which establishes that a contract is executable even though a claim c may not be satisfied, expressed by the constraint Φ_{unsat_c} :

$$\Phi_{unsat\_c} \equiv \phi_{SPA} \wedge c.\texttt{DueDate} = -1. \tag{26}$$

Assume the pretzel bakery example SPA not to specify any restitution claim. When we apply the analysis in Eq. 26 to the `TransferClaim`, the constraint $\Phi^B_{unsat\_TranferClaim}$ will be created:

$$\Phi^B_{unsat\_TranferClaim} \equiv \phi_{SPA} \wedge d_u = -1. \tag{27}$$

Since there is no consequence for the `PerformanceClaim`, no contract execution exists in which this claim is unsatisfied. It follows that $\Phi^B_{unsat\_TranferClaim}$ is unsatisfiable. One possible reason for this analysis to fail is that a primary or independent claim does not possess a consequence claim. Notice that, if for instance a warranty is unsatisfied, a consequence like a compensation should be paid. Without a consequence claim, the encoding enforces that the claim is always satisfied as there is no consequence claim that can be satisfied instead.

Claim defense analysis A claim c can be defensed because of the lack of performance of another claim d. This relation is represented by a *Precede* association. When c would become due before d and d precedes c, then c cannot be performed in time. The analysis of the constraint $\Phi_{Precede\_c\_d}$ checks for such timing inconsistencies. It is satisfiable if the inconsistency exists in the considered SPA:

$$\Phi_{Precede\_c\_d} \equiv \phi_{owner} \wedge \phi_c \wedge \phi_d \wedge c.\texttt{DueDate} < d.\texttt{DueDate}. \tag{28}$$

In the pretzel bakery SPA example, the payment of the bakery precedes the transfer occurring first. Consequently, the due date of the `PayClaim` should not be after the

due date of the TransferClaim, which is tested by checking the satisfiability of the formula $\Phi_{Precede\_Transfer\_Pay}$:

$$\Phi_{Precede\_Transfer\_Pay} \equiv \phi_{owner} \wedge \phi_{Pay} \wedge \phi_{Transfer} \wedge d_{Transfer} < d_{Pay}. \tag{29}$$

Applying the analysis to the pretzel bakery SPA example, the constraint which indicates that there is no such inconsistency between the TransferClaim and the PayClaim. By changing the due date of the payment to 29, $\Phi_{Precede\_Transfer\_Pay}$ becomes satisfiable with $d_{Transfer} = 28$ and $d_{Pay} = 29$, which means that an inconsistency has been detected.

Limitation analysis The *due date* and *limitation* of a claim can refer to other time events in the contract. In the pretzel bakery SPA, for instance, the DueDate of Claim1 is +28. The relative time reference means that Claim1 has to be fulfilled within 28 days after the assert date d_g of PretzelWarranty.

The combination of several *due dates* and *limitations* can result in an inconsistency where, because of its *limitation* date, the *due date* of a claim is later than the date at which the claim expires. Analysis $\Phi_{Limitation\_c}$ checks this inconsistency for every *claim* $c \in \mathcal{C}$ with a Limitation. For this analysis, we need to (textually) substitute $c.\texttt{DueDate} \leq d_c \leq c.\texttt{Limitation}$ by $c.\texttt{DueDate} \leq d_c$, which we indicate by the notation $c.\texttt{DueDate} \leq d_c / c.\texttt{DueDate}$ and the enclosure of the formula to which the substitution applies in squared backets:

$$\begin{aligned}\Phi_{Limitation\_c} \equiv \phi_{SPA}[c.\texttt{DueDate} \leq d_c / c.\texttt{DueDate} \leq d_c \leq c.\texttt{Limitation}] \\ \wedge\ c.Limitation < c.DueDate.\end{aligned} \tag{30}$$

In the bakery SPA, only the claims Claim1 and Claim2 contain a Trigger to PretzelWarranty and a Limitation of 70 days. Hence, the limitation analysis is only created for these two claims. The SMT solver Z3, invoked by *ContractCheck*, computes that $\Phi_{Limitation\_Claim1}$ is unsatisfiable, which entails that the time constraints are consistent. For $\Phi_{Limitation\_Claim2}$, Z3 computes a model that, for instance, contains the following assignments: $d_{Claim1} = -1$, $d_{PretzelWarranty} = 30$ and $d_{Claim2} = 71$. With this contract execution, illustrated by the Sequence Diagram in Fig. 7, Chris asserts the PretzelWarranty on day 29, then Eva tries to perform Claim1 for 28 days but fails. Now, Eva has to pay a compensation that is due after another 14 days. In summary, the compensation claim is due after 71 days, even though, due to the Limitation, the legal basis of the claim is outdated after 70 days. This shows that there is an inconsistency in the timing between the due dates and the Limitation of the PretzelWarranty.

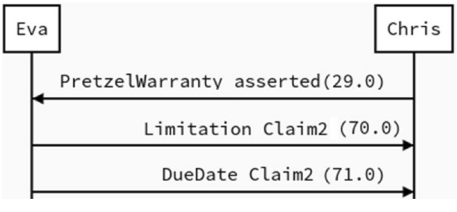


Fig. 7 Sequence diagram computed by *ContractCheck* illustrating a limitation error

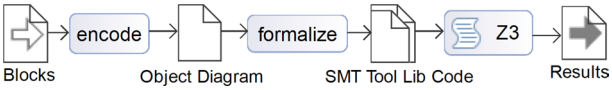


Fig. 8 *ContractCheck* analysis workflow, modeled using the BPMN notation (Object Management Group 2014)

6 The *ContractCheck* tool

6.1 *ContractCheck* workflow

An SPA is usually formulated in natural language and not formalized. Natural Language Processing to automatically obtain a semantic model of the SPA is beyond the scope of this paper. Instead, we provide a collection of parameterized structured English text blocks to the user that can be selected and properly parameterized in order to reflect the meaning of the SPA to be analyzed. The *ContractCheck* (CCT, 2024) tool for the consistency analysis of SPAs that we describe in this section parses these text blocks and translates them into an internal formal representation. In doing so, it gives these text blocks a formal semantics. This relieves the user from the task to provide a semantics, and thereby a formalization, in a manual fashion.

The workflow of the tool *ContractCheck* is depicted in the diagram in Fig. 8. After the user has selected and parameterized the text blocks, *ContractCheck* will compile a UML object diagram from the text blocks and the information regarding the class structure encoded in the SPA class diagram (see Fig. 3). Note that the object diagram merely serves as an internal representation and does not need to be edited by the user. *ContractCheck* extracts the formal representation of the SPA to be analyzed from the object diagram and synthesizes the logical constraints that we presented in Sect. 4 in *smtlib2*-format. Next, *ContractCheck* invokes the Z3 SMT solver to perform the consistency analyses described in Sect. 5. Finally, *ContractCheck* translates the analysis results computed by Z3 into user-readable format and produces the results in its user interface.

6.2 Text blocks in *ContractCheck*

In *ContractCheck*, text blocks serve to capture the semantics of the different clauses in an SPA in a structured fashion. Figure 9 gives an excerpt of the text block that

ID:	Block1
Text:	The seller $seller.Name hereby sells shares of $shares.Name, with all rights and obligations pertaining thereto, to the Purchaser $purchaser.Name, who accepts such sale.
Object:	“spa:SPA”, “seller:Person”,“purchaser:Person”, “share:Share”, “transfer:PrimaryClaim”
Assignment:	“seller.name=Eva”, “purchaser.name=Chris”, “spa.Seller=$seller”, “transfer.Performance=$shares.transfer($purchaser)”, ...

Fig. 9 Excerpt from text block encoding of Bakery SPA in JSON format

represents a portion of the clause 1.1 of the pretzel bakery SPA that can be found in full length in the "Appendix". A text block has a unique identifier (*ID*) that allows one to refer to it. The *Text* field contains a parameterized structured natural language description of the legal content of the contract clause that the text block represents. The *Object* field provides references to the objects that the text block is parameterized with. The *Assignment* field gives concrete values for the object parameters that are referenced in the text block. The information contained in the *Object* and *Assignment* fields is then used by *ContractCheck* to compile the object diagram used as an internal semi-formal representation of the SPA, c.f. Fig. 5.

More concretely, the text block with the ID *Block1* given in Fig. 9 defines the essential components of an SPA: A seller, a purchaser, the shares to be sold and a price. The $-character in the text indicates the assignment of a variable value, such as the attribute `Name` for a person. For instance, the assignment to *$seller.name* is currently *Eva*, as defined in the *Assignment* section of the block. A block may reference the variables of another block by using the $-character. For instance, $Block1_share refers to the variable *share* in *Block1*. The instance *Bakery* of type `Shares` is defined in another block. The inter-block reference semantics is necessary to assign $Block1_share to the attribute `Property` of *Bakery*.

We envision two usage scenarios for *ContractCheck*, see also Fig. 1. One scenario is based on the idea that a comprehensive library of text blocks will be developed, so that existing textual SPAs can be manually mapped to blocks chosen from this library. The parameters of the text blocks will then be filled with concrete values from the textual contract at the time when this mapping is performed. This modeling step is supported by the user interface of *ContractCheck*. The second scenario is based on the idea that the contract will be defined using a set of text blocks from the library that represents the clauses and stipulations of the SPA. A contract text can then be synthesized by *ContractCheck* as a by-product of the analysis. This second scenario fits well into the increasingly popular use of contract synthesis or generation tools in legal practice. The consistency analysis performed by *ContractCheck* is agnostic with respect to the use of either of these two usage scenarios.

6.3 *ContractCheck* analysis result presentation

The analysis results are displayed in the user interface of *ContractCheck*. The results of the Analyses I and II and the further dynamic analyses are either satisfying models, which provide value assignments to the logical variables which render the analysis problem satisfiable, or unsatisfiability cores which give value assignments that render the analysis problem unsatisfiable.

In case a satisfying model is returned, the valuations of the variables in the model indicate possible dates in which claims in the contract are executed. *ContractCheck* depicts such a satisfying model by a sequence diagram. For the bakery SPA, a computed execution is depicted in Fig. 10. It indicates that the `TransferClaim` is unperformed by Eva, the `PretzelWarranty` is performed by her, Chris performs the `PayClaim` on day 28 after signing, and is compensated for this payment by a `Restitution` on day 29. This execution scenario illustrates that the contract

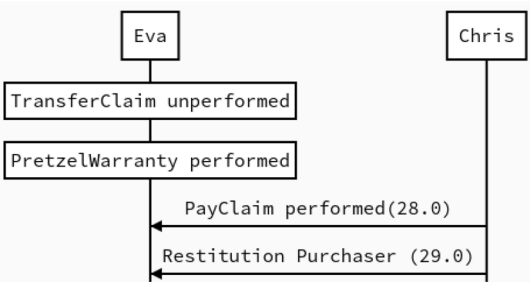


Fig. 10 Bakery SPA execution visualized in the form of a UML sequence diagram

can be executed in spite of the inconsistency regarding the ownership of the bakery, which means that the primary claims cannot be executed.

In case the SMT analysis invoked by *ContractCheck* returns a result of unsatisfiability of the constraint system, an unsatisfiability core will be returned. It contains a maximum number of conjunctively connected logical constraints so that any true subset of it is still satisfiable. The constraints contained in the unsatisfiability core are associated with certain text blocks, which *ContractCheck* "red flags" and draws side by side in the user interface, as shown in Fig. 11. In this case, the *ContractCheck* analysis result reveals the contradiction between the claimed pretzel bakery ownership by Eva in Block1, and the stated transfer of the pretzel bakery ownership to the bank in Block 11. These are the text blocks that created a set of unsatisfiable constraints contained in the unsatisfiability core of $\Phi_{TransferClaim}$. The depicted claims help the user to identify the claims that contradict each other and, hence, to "debug" the considered SPA.

In order to test our approach, we created an instance of the Bakery example without inconsistencies. Namely, the bank is no longer owner of the bakery and we also set the warranty to a limitation of 56 days. The tool does not find any inconsistency and depicts the execution in Fig. 12.

Fig. 11 Inconsistent output for the Bakery SPA

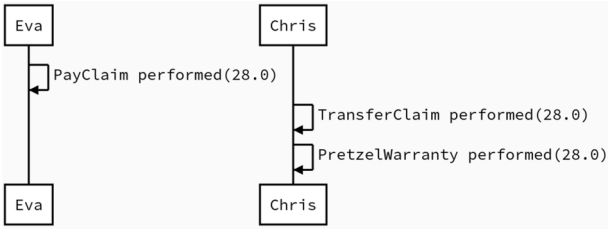


Fig. 12 Execution of the Bakery SPA without inconsistencies

7 Case study

To evaluate *ContractCheck* on an SPA of realistic size we have applied it to a slightly modified version of an SPA-template taken from Meyer-Sparenberg (2022b). This contract template is bilingual in German and English. It consists of twelve paragraphs. The combined German and English text spans 21 pages of a Word document and contains about 11.300 words.

We used Meyer-Sparenberg's original template as a starting point and adapted it by incorporating provisions commonly found in a wide range of standard contracts, which yielded the contract text that we use as a case study in this section. To identify these common clauses, we reviewed the sample contracts provided in the form books (von Hoyenberg 2020a, b; Meyer-Sparenberg 2022a, b; Pfisterer 2022; Seibt 2018a; Seibt 2018; Seibt 2018c, d). A quantitative analysis based on matching headings in the case study contract to headings in the form book sample contracts revealed that the clauses included in the case study contract also appear in between 75% and 100% of the contracts provided by the considered form books. It is safe to assume that a large portion of the SPAs used in practice are based on these form book samples, because, to our knowledge, these form books are the most commonly used in practice in Germany. We consequently assume that our case study reflects a wide range of standard clauses and provisions, making it a representative test case for ContractCheck.

In order to be able to analyze the capabilities of *ContractCheck* to detect inconsistencies in contract texts it is necessary to seed inconsistencies into these texts (Jia and Harman 2011). In particular, the following five inconsistencies were seeded:

(a) The contract states that the seller owns 100% of the company shares, but a warranty clause was modified to state that the seller owns only 90% of the shares. This creates an inconsistency between the actual share ownership specified and what is warranted.
(b) The minimal compensation of a compensation claim is 20.000€. The sum of all compensations is restricted by a limitation of liabilities to 1% of the purchase price of 1.800.000. As a consequence, no compensation can ever be paid.
(c) A clause was added stating that non-existing real estate properties used by the company have no third party rights. However, the contract specifies that the company owns no real estate property, so this clause is inconsistent.
(d) The clause regulating payment of an additional purchase price upon adoption of the company's annual financial statements was modified to remove the requirement that the statements be transmitted to the seller. This creates a potential inconsistency where payment could become due before the seller receives the statements.
(e) The effective date for accrual of interest on the purchase price was falsely specified as a date that comes after the stipulated latest possible closing date. This results in an impossibility where interest would accrue after the date the price is paid.

Table 1 Quantitative experimental results for the analysis of a realistic size contract

Analysis	*#Err/#An*	Time	Memory	*#Var*	*#Con*
Contract Executability (II)	0/ 1	0.030s	3.17MB	141	491
Claim Consistency (I)	5/47	0.005s	1.45MB	21	48
Claim Unsatisfiable	1/47	0.022s	3.02MB	138	312
Claim Defense	1/ 1	0.002s	1.32MB	19	30
Limitation Check	1/13	0.017s	3.16MB	141	495

Maximal result values for individual analyses in each of the analysis categories (rows)

As we shall illustrate next, all of these seeded faults can be detected by *ContractCheck* using one of the defined semantic consistency analyses.

Analysis results We have applied the analyses implemented in *LegalCheck* to the inconsistency-seeded realistic size SPA contract described above. From a qualitative perspective, all seeded inconsistencies were detected by *LegalCheck*.

An overview of the quantitative results can be seen in Table 1. For the different types of analyses mentioned in column *Analysis*, *ContractCheck* created a number *#An* of SMT analyses, and found a number *#Err* of errors in the SPA. A particular type of consistency analysis encoded as an SMT problem contains at most a number *#Var* of variables and *#Con* of constraints. In other words, these two columns indicate the maximum size of a constraint system solved by the SMT solver in any problem in a particular row of the table. For each type of analysis, the column *Time* indicates the maximum computation time and the column *Memory* the maximum amount of consumed main memory that it took to check satisfiability of any problem in that row of the table.

The analysis type *Contract Executability (Analysis II)* returned one execution of the SPA which shows that the SPA is executable. The analysis type *Claim Consistency (Analysis I)* created 47 SMT satisfiability analysis problems and succeeded in detecting the seeded error *a* and 4 compensation claims that can never be paid because of the seeded inconsistency *b*. The analysis type *Claim Unsatisfiable* created 47 SMT satisfiability analysis problems and succeeded in detecting the seeded error *c*. The analysis type *Claim Defense* created 1 SMT satisfiability analysis problems and succeeded in detecting the seeded error *d*. The analysis type *Limitation Check* created 13 SMT satisfiability analysis problems and succeeded in detecting the seeded error *e*. For every analysis, the memory demand of the SMT-solver was below 3.17MB, and satisfiability was decided within at most 30ms. These results show that our encoding of the logical conditions allows for an efficient SMT-based satisfiability analysis for SPAs of realistic size.

In conclusion, the case study shows that the proposed method is effective, i.e., that it finds inconsistencies in a more complex contract, and that is its efficient, i.e., that the formalization and the SMT solving scale to a contract of more realistic size and complexity.

8 Threats to validity

We view the following points as main threats to the validity of our results:

- It is possible that there are mismatches between the intended semantics of a contract text and the semantics entailed by our formalization. We believe that such a mismatch is rather unlikely since the formalizations of the different types of claims have been carefully reviewed by the legal experts in the author team of this paper.
- There is a chance that the reasoning engine used in our paper is inconsistent. Again, the chances of this happening are rather low since the Z3 SMT solver that we use is in heavy academic and industrial use, for instance within Microsoft Corp. (Bjørner 2018), and can consequently be considered to be well tested and stable.
- It may be that the set of blocks that we propose in this paper does not correspond well to the types of stipulations encountered in practical use, i.e., that our blocks do not generalize well. As we argue in Sect. 7, the clauses used in our case study of complex contracts generalize well to a large set of sample contracts from form books widely used in practice in Germany. We maintain that this implies that a fair number of blocks that we developed for our case studies can be reused in the formalization of a large number of SPAs that can be found in practice.
- At this point it is unknown which portion of the types of stipulations that occur in practical contracts are covered by the blocks for which we have proposed a formalization and an analysis. We assume that over time a large library of reusable blocks will be developed so that coverage will steadily increase. Notice that we do not claim completeness in the sense that all inconsistencies in a given SPA contract text will be identified – we merely stipulate that our analysis is complete with respect to the set of blocks used in the analysis of a given contract and the formalizations proposed for this set of blocks.

9 Conclusion

We have presented a method for the logical modeling and consistency analysis of legal contracts using an SPA as an example. We provided an ontology for SPAs using UML class diagrams and illustrated the refinement of this ontology to a UML object diagram as a case study. We discussed the logical formalization of the SPA using decidable fragments of FOL via SMT solving. Finally, we introduced the tool *ContractCheck*, which allows the textual editing of contracts using building blocks, performs the automated derivation of the logical encoding of the contract and the consistency conditions, invokes the Z3 SMT solver, and returns the analysis results to the user. We have evaluated the consistency analyses that *ContractCheck* can perform on an SPA of realistic size and noted that significant seeded inconsistencies in the contract could be discovered.

Future research will increase the size and complexity of the contract artifacts that we consider. We will further develop the analysis of the dynamic execution of contracts by introducing state-machine models, among other things, in order to assess the benefits of the contract for different contracting parties in the light of the possible dynamic execution scenarios. We are also currently working on an approach relying on machine-learning based Natural Language Processing that automatically maps natural language SPA texts to the blocks that we describe in this paper.

We see this work as an innovative contribution to improving the quality of complex legal artifacts using logic-based, automated analysis methods.

Appendix

Text of the Pretzel Bakery SPA

§1 **Main Content**

1.1 The Seller Eva hereby sells the shares of Bakery AG with all rights and obligations pertaining thereto (including the dividend right for the current financial year), to the Purchaser Chris who accepts such sale.
1.2 The Purchaser pays the purchase price 40.000€ to the Seller.
1.3 If the transfer is not performed, the Purchaser has the right to withdraw.
1.4 If the payment is not performed, the Seller has the right to withdraw.

§2 The Seller hereby represents and warrants to the Purchaser in the form of an independent guarantee pursuant to Section 311 (1) of the German Civil Code and exclusively in accordance with the provisions of this Agreement that the following statements (the "Warranties") are true and correct as of the date of this agreement and that the Warranties set forth in this paragraph will also be true and correct as of the Closing Date:

2.1 The company can produce at least the 10.000 of Pretzels every day (Pretzel Warranty). In case of the breach of the Warranty, it needs to be asserted within 14 days.

§3 The Purchaser's rights arising from any inaccuracy of any of the Warranties contained in §1 shall be limited to supplementary performance claims and compensation claims against the Seller, subject to the provisions of

3.1 In case the Pretzel Warranty is not met and then the creditor may demand subsequent performance within 28 business days from the debtor after having transfered the shares.
3.2 In case the Pretzel Warranty is not met and the damage is above 1.000€ then a compensation of 100€ per 100 pretzels not baked pretzels is paid within 14 business days.

§4 Claims of §3 expire after 42 business days.

§5 The Bakery AG is transferred by way of security to Bank B.

Text blocks for the Pretzel Bakery SPA

ID:	Block1
Text:	The seller $seller.Name hereby sells shares of $shares.Name, with all rights and obligations pertaining thereto (including the dividend right for the current financial year), to the Purchaser $purchaser.Name, who accepts such sale.
Object:	"spa:SPA", "seller:Person", "purchaser:Person", "shares:Shares", "transfer:PrimaryClaim"
Assignment:	"purchaser.Name=Chris", "seller.Name=Eva", "spa.Seller=$seller", "spa.Purchaser=$purchaser", "shares.Name=Bakery AG", "spa.Object=$shares", "spa.Claim=$transfer", "spa.Closing=28", "transfer.Performance=Bakery.transfer($purchaser)", "transfer.Debtor=$seller", "transfer.Creditor=$purchaser", "transfer.DueDate=28"
ID:	Block2
Text:	The purchaser pays the purchase price $price.Amount € to the seller on date $payment.DueDate.
Object:	"spa:$SPA", "price:PurchasePrice", "payment:PrimaryClaim"
Assignment:	"spa=$Block1_spa", "spa.Price=$price", "price.Amount=40000", "payment.Debtor=Block1_Purchaser", "spa.Claim=$payment", "payment.Creditor=Block1_Seller", "payment.DueDate=28", "payment.Performance=price.transfer($seller)"
ID:	Block3
Text:	If the $claim is not performed, the $withdraw.Creditor has the right to withdraw.
Object:	"claim:$Claim", "withdraw:RestitutionClaim"
Assignment:	"claim=$Block1_transfer", "withdraw.Name=Restitution Purchaser", "withdraw.Trigger=$claim", "withdraw.Debtor=$claim.Creditor", "withdraw.Creditor=$claim.Debtor"
ID:	Block4
Text:	If the $claim is not performed, the $withdraw.Creditor has the right to withdraw.
Object:	"claim:$Claim", "withdraw:RestitutionClaim"
Assignment:	"claim=$Block2_payment", "withdraw.Name=Restitution Seller", "withdraw.Trigger=$claim", "withdraw.Debtor=$claim.Creditor", "withdraw.Creditor=$claim.Debtor"
ID:	Block5
Text:	The Seller hereby represents and warrants to the Purchaser in the form of an independent guarantee pursuant to Section 311 (1) of the German Civil Code and exclusively in accordance with the provisions of this Agreement that the following statements (the "Warranties") are true and correct as of the date of this Agreement and that the Warranties set forth in this paragraph will also be true and correct as of the Closing Date:
ID:	Block6
Text:	The company can produce at least the $amount of $thing every day. In case of the breach of the warranty, it needs to be asserted within $warranty.Limitation days.
Object:	"warranty:WarrantyClaim", "count:Integer", "amount:Integer", "thing:String"
Assignment:	"warranty.Name=PretzelWarranty", "warranty.Debtor=$Block1_seller", "warranty.DueDate=$Block1_spa.Closing", "thing=Pretzels", "warranty.Creditor=$Block1_purchaser", "warranty.Limitation = +14", "warranty.Performance=(Block6_count=Block6_amount)", "amount=10000", "Block1_spa.Claim=$warranty"
ID:	Block7

ID:	Block1
Text:	The Purchaser's rights arising from any inaccuracy of any of the Warranties contained in $block shall be limited to supplementary performance claims and compensation claims against the Seller, subject to the provisions of
Object:	"claim:$Claim", "per:PerformanceClaim", "block:$Block"
Assignment:	"block=$Block6", "claim=$Block6_warranty", "per.Trigger=$claim", "per.Name=Claim1", "per.DueDate=+28", "per.Debtor=$claim.Debtor", "per.Creditor=$claim.Creditor"
ID:	Block8
Text:	In case the $claim is not met and then the creditor may demand subsequent performance within $per.DueDate business days from the debtor after having transfered the shares.
Object:	"claim:$Claim", "per:PerformanceClaim"
Assignment:	"claim=$Block6_warranty", "per.Name=Claim1", "per.Trigger=$claim", "per.DueDate=+28", "per.Performance=$claim.Performance", "per.Debtor=$claim.Debtor", "per.Creditor=$claim.Creditor"
ID:	Block9
Text:	In case the $claim is not met and the damage is above $comp.Min €then a compensation $claim.Performance is paid within $comp.DueDate days.
Object:	"claim:$Claim", "comp:CompensationClaim"
Assignment:	"claim=$Block6_warranty", "comp.Name=Claim2", "comp.Min=1000", "comp.DueDate=+42","comp.Trigger=$claim", "comp.Compensation=((Block6_amount-Block6_count)/100)*1000", "comp.Debtor=$claim.Debtor", "comp.Creditor=$claim.Creditor"
ID:	Block10
Text:	Claims in $block expire after $d business days.
Objects:	"claim:$Claim", "d:Date", "block:Block"
Assignment:	"block=Block8", "d=28+42", "${//$block//Claim}.Limitation=$d"
ID:	Block11
Text:	The $object is transferred by way of security to $owner.Name.
Objects:	"owner:Person", "object:$Object", "prop:PropertyRight"
Assignment:	"owner.Name=Bank", "object=$Block1_shares", "prop.Owner=$owner", "prop.Property=$object"

Acknowledgements We wish to thank Raffael Senn and David Boetius for helpful comments on an earlier version of this paper.

Author contribution Martin Kölbl: The majority of this author's contributions were made while affiliated with the University of Konstanz.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest None of the authors has any conflict of interest related to the research results documented in this manuscript. No extra-mural funds, grants, or other support was received. The research was entirely funded by government operating funds of the Unviersity of Konstanz. The code of the *ContractCheck* software tool is publicly available, as cited in the manuscript.

References

(2025) Bürgerliches Gesetzbuch, German Civil Code

(2024) ContractCheck. https://github.com/sen-uni-kn/ContractCheck

Alpern B, Schneider FB (1987) Recognizing safety and liveness. Distrib Comput 2(3):117–126. https://doi.org/10.1007/BF01782772

Balbiani P, Broersen JM, Brunel J (2009) Decision procedures for a deontic logic modeling temporal inheritance of obligations. Electron Notes Theor Comput Sci 231:69–89. https://doi.org/10.1016/j.entcs.2009.02.030

Barrett C, Fontaine P, Tinelli C (2021) The SMT-LIB standard version 2.6. https://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2021-05-12.pdf

Bjørner NS (2009) SMT solvers for testing, program analysis and verification at microsoft. In: SYNASC. IEEE Computer Society, p 15

Bjørner NS (2018) Z3 and SMT in industrial R &D. In: FM, Lecture notes in computer science, vol 10951. Springer, pp 675–678

Boley H, Paschke A, Shafiq MO (2010) Ruleml 1.0: the overarching specification of web rules. In: Dean M, Hall J, Rotolo A et al (eds) RuleML 2010: semantic web rules. International symposium. Springer, pp 162–178. https://doi.org/10.1007/978-3-642-16289-3_15

Bonifácio AL, Mura W (2021) Automatically running experiments on checking multi-party contracts. Artif Intell Law 29(3):287–310. https://doi.org/10.1007/s10506-020-09276-y

Braegelmann T, Kaulartz M (2019) Rechtshandbuch smart contracts. C.H. Beck

Camilleri JJ, Schneider G (2017) Modelling and analysis of normative documents. J Log Algebraic Methods Program 91:33–59. https://doi.org/10.1016/j.jlamp.2017.05.002

Castro PF, Maibaum TSE (2007) A complete and compact propositional deontic logic. In: Jones CB, Liu Z, Woodcock J (eds) ICTAC 2007: theoretical aspects of computing. Springer, pp 109–123. https://doi.org/10.1007/978-3-540-75292-9_8

Castro PF, Maibaum TSE (2008) A tableaux system for deontic action logic. In: Meyden R, Torre L (eds) DEON 2008: Deontic logic in computer science. 9th international conference. Springer, pp 34–48. https://doi.org/10.1007/978-3-540-70525-3_4

Clarke EM, Henzinger TA, Veith H et al (2018) Handbook of model checking. Springer. https://doi.org/10.1007/978-3-319-10575-8

Coates JC (2016) M &A contracts: purposes, types, regulation, and patterns of practice. In: Hill CA, Solomon SD (eds) Research handbook on mergers and acquisitions, pp 29–65

Crafa S, Laneve C, Sartor G et al (2023) Pacta sunt servanda: legal contracts in stipula. Sci Comput Program 225(102):911. https://doi.org/10.1016/j.scico.2022.102911

de Moura LM, Bjørner N (2008) Z3: an efficient SMT solver. In: TACAS, Lecture notes in computer science, vol 4963. Springer, pp 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

Desai N, Narendra NC, Singh MP (2008) Checking correctness of business contracts via commitments. In: Padgham L, Parkes D, Müller J et al (eds) AAMAS '08: proceedings of the 7th international joint conference on autonomous agents and multiagent systems, vol 2, pp 787–794. https://doi.org/10.5555/1402298.1402333

Engers T, Gerrits R, Boekenoogen M et al (2001) POWER: using UML/OCL for modeling legislation—an application report. In: 8th international conference on artificial intelligence and law. Association for Computing Machinery, pp 157–167. https://doi.org/10.1145/383535.383554

Faciano C, Mera S, Schapachnik F et al (2017) Performance improvement on legal model checking. In: Governatori G, Keppens J (eds) ICAIL '17: proceedings of the 16th edition of the international conference on articial intelligence and law. ACM, pp 59–68. https://doi.org/10.1145/3086512.3086518

Gorín D, Mera S, Schapachnik F (2010) Model checking legal documents. In: JURIX, Frontiers in artificial intelligence and applications, vol 223. IOS Press, pp 151–154. https://doi.org/10.5555/1940559.1940582

Gorín D, Mera S, Schapachnik F (2011) A software tool for legal drafting. In: FLACOS, pp 71–86. https://doi.org/10.4204/EPTCS.68.7

Governatori G, Wong M (2023) Defeasible semantics for l4. https://doi.org/10.13140/RG.2.2.27931.34089

Grupp M (2018) Wie baut man einen Rechtsautomaten? In: Hartung M, Bues MM, Halbleib G (eds) Legal Technique. C.H. Beck, pp 259–266

Henglein F, Larsen CK, Murawska A (2020) A formally verified static analysis framework for compositional contracts. In: Financial cryptography workshops, lecture notes in computer science, vol 12063. Springer, pp 599–619. https://doi.org/10.1007/978-3-030-54455-3_42

Hildebrandt T, Mukkamala RR, Slaats T (2012) Nested dynamic condition response graphs. In: Arbab F, Sirjani M (eds) Fundamentals of software engineering. 4th IPM international conference, FSEN 2011, pp 343–350. https://doi.org/10.1007/978-3-642-29320-7_23

Hvitved T, Klaedtke F, Zalinescu E (2012) A trace-based model for multiparty contracts. J Log Algebraic Methods Program 81(2):72–98. https://doi.org/10.1016/j.jlap.2011.04.010

Jia Y, Harman M (2011) An analysis and survey of the development of mutation testing. IEEE Trans Software Eng 37(5):649–678. https://doi.org/10.1109/TSE.2010.62

Kabilan V (2005) Contract workflow model patterns using BPMN. In: Halpin TA, Siau K, Krogstie J (eds) Proceedings of the 10th international workshop on exploring modeling methods for systems analysis and design, EMMSAD 2005, pp 171–182

Kabilan V, Johannesson P (2003) Semantic representation of contract knowledge using multi tier ontology. In: Cruz IF, Kashyap V, Decker S et al (eds) Proceedings of SWDB'03, the first international workshop on semantic web and databases, co-located with VLDB 2003. Humboldt-Universität, Berlin, Germany, September 7–8, 2003, pp 395–414. https://doi.org/10.5555/2889905.2889930

Kennedy D, Fisher WW (eds) (2007) Wesley Hohfeld, some fundamental legal conceptions as applied in judicial reasoning, 23 Yale Law Journal 16 (1913), Princeton University Press, Princeton, pp 45–82. https://doi.org/10.1515/9780691186429-004

Khoja A, Kölbl M, Leue S et al (2022a) Automated consistency analysis for legal contracts. In: SPIN, Lecture notes in computer science, vol 13255. Springer, pp 1–23. https://doi.org/10.1007/978-3-031-15077-7_1

Khoja A, Kölbl M, Leue S et al (2022b) Formal modeling and analysis of legal contracts using contractcheck. arXiv: org/abs/2212.03349

Kroening D, Strichman O (2016) Decision procedures—an algorithmic point of view, 2nd edn. Texts in theoretical computer science. An EATCS series. Springer. https://doi.org/10.1007/978-3-662-50497-0

Madaan N, Krishna PR, Karlapalem K (2014) Consistency detection in e-contract documents. In: ICEGOV. ACM, pp 267–274. https://doi.org/10.1145/2691195.2691249

Martínez E, Díaz G, Cambronero M et al (2010) A model for visual specification of e-contracts. In: IEEE SCC. IEEE Computer Society, pp 1–8. https://doi.org/10.1109/SCC.2010.32

McNamara P, Van De Putte F (2022) Deontic logic. In: Zalta EN, Nodelman U (eds) The stanford encyclopedia of philosophy, fall, 2022nd edn. Stanford University, Metaphysics Research Lab

Merigoux D, Chataing N, Protzenko J (2021) Catala: a programming language for the law. In: ICFP 2021. https://doi.org/10.1145/3473582

Meyer-Sparenberg W (2022a) Form. iii. a. 16. unternehmenskaufvertrag (gmbh-anteile) – käuferfreundlich. In: Gebele A, Scholz KS (eds) Beck'sches Formularbuch Bürgerliches, Handels- und Wirtschaftsrecht. C.H. Beck

Meyer-Sparenberg W (2022b) Form. iii. a. 17. unternehmenskaufvertrag (gmbh-anteile) – verkäuferfreundlich. In: Gebele A, Scholz KS (eds) Beck'sches Formularbuch Bürgerliches, Handels- und Wirtschaftsrecht. C.H. Beck

Mitra S, Anand K, Chittimalli PK (2018) Identifying anomalies in sbvr-based business rules using directed graphs and smt-libv2. In: ICEIS (2), pp 215–222

Mura WAD, Bonifácio AL (2015) Devising a conflict detection method for multi-party contracts. In: SCCC. IEEE, pp 1–6. https://doi.org/10.1109/SCCC.2015.7416574

OASIS Standard (2021) LegalRuleML, version 1.0. https://docs.oasis-open.org/legalruleml/legalruleml-core-spec/v1.0/os/legalruleml-core-spec-v1.0-os.pdf

Object Management Group (2014) Business process model and notation. https://www.omg.org/spec/BPMN

Object Management Group (2017) Unified modelling language, Specification 2.5.1. http://www.omg.org/spec/UML

Oetker H, Maultzsch F (2018) Vertragliche schuldverhältnisse, 5th edn. Springer, Berlin, pp 17–301

Paal BP, Fries M (2019) Smart contracts. Mohr Siebeck

Pace GJ, Prisacariu C, Schneider G (2007) Model checking contracts—a case study. In: ATVA, Lecture notes in computer science, vol 4762. Springer, pp 82–97. https://doi.org/10.1007/978-3-540-75596-8_8

Palmirani M, Governatori G, Rotolo A et al (2011) Legalruleml: Xml-based rules and norms. In: RuleML America, Lecture notes in computer science, vol 7018. Springer, pp 298–312. https://doi.org/10.1007/978-3-642-24908-2_30

Pfisterer B (2022) Share deal. In: Weise S, Krauß HF (eds) Beck'sche Online-Formulare. C.H. Beck

Prisacariu C, Schneider G (2012) A dynamic deontic logic for complex contracts. J Log Algebraic Methods Program 81(4):458–490. https://doi.org/10.1016/j.jlap.2012.03.003

Prisacariu C, Schneider G (2007) A formal language for electronic contracts. In: FMOODS, Lecture notes in computer science, vol 4468. Springer, pp 174–189. https://doi.org/10.1007/978-3-540-72952-5_11

Schumann G, Gómex JM (2024) Detection of conflicts, contradictions and inconsistencies in regulatory documents: a literature review. In: 2024 Fifth international conference on intelligent data science technologies and applications (IDSTA). IEEE, pp 81–88

Seibt CH (2018a) GmbH-Anteilskaufvertrag - ausführlich, käuferfreundlich. In: Beck'sches Formularbuch Mergers & acquisitions. C.H. Beck, pp 324–456

Seibt CH (2018c) GmbH-Anteilskaufvertrag - knapp, ausgewogen. In: Beck'sches Formularbuch Mergers & Acquisitions. C.H. Beck, pp 515–525

Seibt CH (2018d) GmbH-Anteilskaufvertrag - knapp, verkäuferfreundlich. In: Beck'sches Formularbuch Mergers & Acquisitions. C.H. Beck, pp 457–514

Seibt CH (2018) GmbH-Anteilskaufvertrag - ausführlich, verkäuferfreundlich, deutsch. Beck'sches Formularbuch Mergers & Acquisitions. C.H. Beck, München, pp 233–323

van Binsbergen LT, Liu LC, van Doesburg R et al (2020) eFLINT: a domain-specific language for executable norm specifications. In: Erwig M, Gray J (eds) Proceedings of the 19th ACM SIGPLAN international conference on generative programming: concepts and experiences, pp 124–136. https://doi.org/10.1145/3425898.3426958

Van Doesburg R, Van Der Storm T, Van Engers T (2016) Calculemus: towards a formal language for the interpretation of normative systems. Artif Intell Justice 1:73

von Hoyenberg P (2020) Share deal (GmbH, fester Kaufpreis). In: Weipert L, Arnhold P, Baltus M (eds) Münchener Vertragshandbuch. C.H. Beck, pp 228–233

von Hoyenberg P (2020) Share deal (GmbH, mit Stichtagsbilanzierung). In: Weipert L, Arnhold P, Baltus M (eds) Münchener Vertragshandbuch. Beck-online Bücher. C.H. Beck, pp 203–227

Von Wright GH (1951) Deontic logic. Mind 60(237):1–15. https://doi.org/10.1093/mind/LX.237.1

Wan F, Singh MP (2005) Formalizing and achieving multiparty agreements via commitments. In: AAMAS, pp 770–777. https://doi.org/10.1145/1082473.1082591

Wilhelmi R (2023) Vor §903. In: Westermann HP, Grunewald B, Maier-Raimer G (eds) Erman BGB. Otto Schmidt

Wilhelmi R (2024a) §453. In: Gsell B, Krüger W, Lorenz S et al (eds) Beck'scher Online Großkommentar. C.H. Beck, pp 290–295

Wilhelmi R (2024b) §453. In: Gsell B, Krüger W, Lorenz S et al (eds) Beck'scher Online Großkommentar. C.H. Beck, pp 505–514

Wilhelmi R (2024c) §453. In: Gsell B, Krüger W, Lorenz S et al (eds) Beck'scher Online Großkommentar. C.H. Beck, pp 744–782